UNIVERSITÄT KOBLENZ · LANDAU

ist

Studienarbeit

Webbasiertes Studiengang-Informationssystem

8. Februar 2008

vorgelegt von

Christian Bruckhoff <brchrist@uni-koblenz.de>
José Angel Monte Barreto <monte@uni-koblenz.de>

Betreuer

Prof. Dr. Jürgen Ebert <ebert@uni-koblenz.de>
Daniel Bildhauer <dbildh@uni-koblenz.de>

Erklärung

Wir versichern, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Wir übernehmen die gemeinsame Verantwortung für die gesamte Studienarbeit.

Die nachfolgend genannten Abschnitte wurden hauptsächlich von *José Angel Monte Barreto* verfasst:

- Kapitel 1
- Abschnitt 3.1
- Abschnitt 4.1
- Kapitel 5
- Kapitel 7

Die nachfolgend genannten Abschnitte wurden hauptsächlich von *Christian Bruckhoff* verfasst:

- Abschnitt 2.2
- Abschnitt 2.3
- Abschnitt 3.2
- Abschnitt 4.2
- Kapitel 6

Alle nicht genannten Abschnitte wurden von beiden Studienarbeitern zu etwa gleichen Teilen bearbeitet.

Mit der Einstellung dieser Arbeit in die Bibliothek sind wir einverstanden.
Der Veröffentlichung dieser Arbeit im Internet stimmen wir zu.

Danksagung

Wir möchten uns bei *Prof. Dr. Jürgen Ebert* und *Daniel Bildhauer* für die gute Betreuung, die vielen hilfreichen Ratschläge und eine angenehme Arbeitsatmosphäre bedanken. Weiterhin gilt ein besonderer Dank *Dr. Torsten Gipp*, der uns in der Implementationsphase bei der Integration von WebSIs in Zope sehr engagiert geholfen hat. Ebenso gilt unser Dank *Prof. Dr. Klaus G. Troitzsch* für die Unterstützung bei Problemen bezüglich der Prüfungsordnung und *Rainer Krienke* für die Unterstützung beim Einrichten der Datenbank. Weiterhin möchten wir uns bei den Beta-Testern *Sabine Orth*, *Sascha Rutenbeck*, *Dr. Volker Riediger*, *Dr. Torsten Gipp* und *Prof. Dr. Jürgen Ebert* für ihr Engagement bedanken.

Christian Bruckhoff bedankt sich zudem bei seinem Arbeitspartner *José Angel Monte Barreto* für die Geduld aufgrund der langsamen Fortschritte an der Studienarbeit wegen gesundheitlichen Problemen.

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Ziel dieser Studienarbeit ist die Erstellung einer öffentlich zugänglichen Online-Webpräsenz, welche den Aufbau und die Struktur der neuen Bachelor- und Masterstudiengänge informativ umsetzen soll. Es soll hierfür ein isoliertes, datenbankbasiertes Informationssystem (im Folgenden WebSIs genannt) ohne jegliche Kopplung mit anderen Systemen erstellt werden. Jedoch sollte es (in Zukunft) über eine Export- bzw. Importschnittstelle verfügen, um den strukturierten Datenaustausch mit anderen Systemen zu erlauben. Zudem soll das System in seinem nicht-öffentlichen Bereich der zentralen (Online-)Pflege des Modulhandbuchs des Fachbereichs 4 (Informatik) dienen.

1.2 Bachelor- und Masterstudiengänge

Seit dem Wintersemester 2006/07 bietet der Fachbereich Informatik der Universität Koblenz-Landau die neuen Bachelorstudiengänge Computervisualistik und Informatik an, welche die alten Diplomstudiengänge Informatik und Computervisualistik ablösen. Der bereits in Bachelor-/Masterform angebotene Studiengang Informationsmanagement hat im Rahmen dieser Umstellung eine Restrukturierung erfahren [Uni06a].

Ebenfalls ab dem Wintersemester 2006/07 werden die Masterstudiengänge Informationsmanagement und Wirtschaftsinformatik angeboten. Mit dem Wintersemester 2007/08 kommen die Masterstudiengänge für Computervisualistik und Informatik hinzu [Uni06b].

Anlass der Umstellung ist der sog. *Bologna-Prozess*, welches nach einer Entscheidung der EU-Bildungsminister/innen vom Juni 1999 die vollständige Umsetzung eines einheitlichen europäischen Hochschulraums bis 2010 zur besseren europaweiten Vergleichbarkeit von Hochschulabschlüssen als Primärziel definiert. Dadurch

können z.B. im Ausland erworbene Leistungen einfacher an inländischen Hochschulen anerkannt werden [Bun04].

Ein Studierender an der Universität Koblenz-Landau, welcher nach einer Regelstudienzeit von 3 Jahren (6 Semestern) einen Bachelorabschluss nach abgeschlossenem Bachelorstudium und erfolgreich bestandenen Bachelorprüfungen erwirbt, erhält den Titel *Bachelor of Science*. Er besitzt damit bereits einen berufsqualifizierenden Abschluss, welcher - im Unterschied zum Vordiplom - den Einstieg in den Arbeitsmarkt ermöglicht.

Mit der Grundlage eines erfolgreich abgeschlossenen Bachelorstudiums wird dem Studierenden nach 2 weiteren Studienjahren (4 Semestern) mit abgeschlossenem Masterstudium und erfolgreich bestandenen Masterprüfungen der akademische Grad des *Master of Science* verliehen, welches in etwa dem Diplom entspricht. Der Mastertitel eröffnet dem Studierenden zudem die Möglichkeit zur Promotion [Uni07].

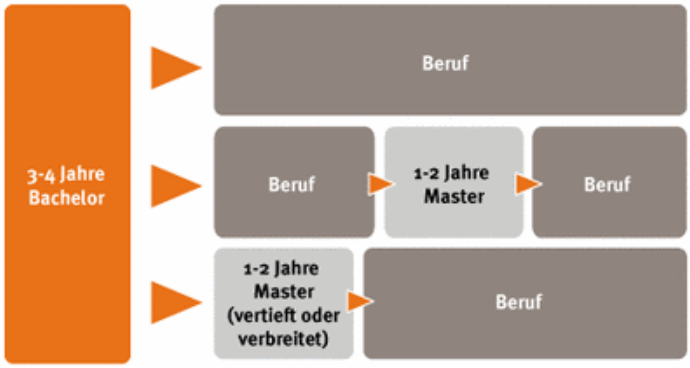

Abbildung 1.1: Studienstruktur der Bachelor- und Masterstudiengänge [Weg07]

Abbildung 1.1 zeigt die drei verschiedenen Varianten der Bachelor- und Masterprogramme, die an deutschen Hochschulen angeboten werden. Demnach stellen das Bachelor- und das Masterstudium zwei getrennte Studiengänge dar. Der Studierende hat nach einem erfolgreich abgeschlossenem Bachelorstudium einerseits die Möglichkeit direkt in den Beruf einzusteigen oder andererseits zunächst seine Fachkenntnisse in einem anschließenden Masterstudium zu vertiefen. Einige Masterstudiengänge werden zudem berufsbegleitend angeboten. An der Universität Koblenz-Landau ist lediglich die konsekutive Variante - sprich Bachelorstudium mit direkt anschließendem Masterstudium - möglich.

Die Lehreinheiten der Bachelor- und Masterstudiengänge werden als *Module* bezeichnet, welche thematisch abgeschlossen sind und wiederum aus einzelnen Lehrveranstaltungen (bspw. Vorlesung, Übung, Seminar etc.) bestehen können. Für jedes Modul wird eine Gesamtnote vergeben. Als Prüfungsleistung werden für die

Module und damit für die darin enthaltenen Lehrveranstaltungen meist Klausuren durchgeführt.

Als Bewertungseinheit werden sog. *Leistungspunkte* oder *Credit Points (CP)* verwendet, welche sich an den Bestimmungen des *European Credit Transfer and Accumulation System* orientieren. Demnach müssen Studenten pro Studienjahr 60, pro Semester 30 Leistungspunkte erwerben. Ein Bachelorstudiengang umfasst demnach 180, ein Masterstudiengang 120 Leistungspunkte. Der Grund für die Verwendung des Leistungspunktesystems ist die Mitberücksichtigung des Eigenstudiums - also die Zeit für die Vor- und Nachbereitung von Veranstaltungen - in Ergänzung zum Präsenzstudium. Im alten System der Semesterwochenstunden (SWS) wurde lediglich letzteres erfasst. Durch das Sammeln von Leistungspunkten werden Leistungsnachweise für den jeweiligen Studienabschluss studienbegleitend erworben, die in ihrer Gesamtheit die Abschlussnote beeinflussen - bei den Diplomstudiengängen wurde diese zum größten Teil von den Ergebnissen der letzten großen Prüfungen bestimmt. 1 Leistungspunkt entspricht dabei ca. 25-30 Arbeitsstunden [Eur06].

2 Anforderungen

2.1 Anwendungsfälle

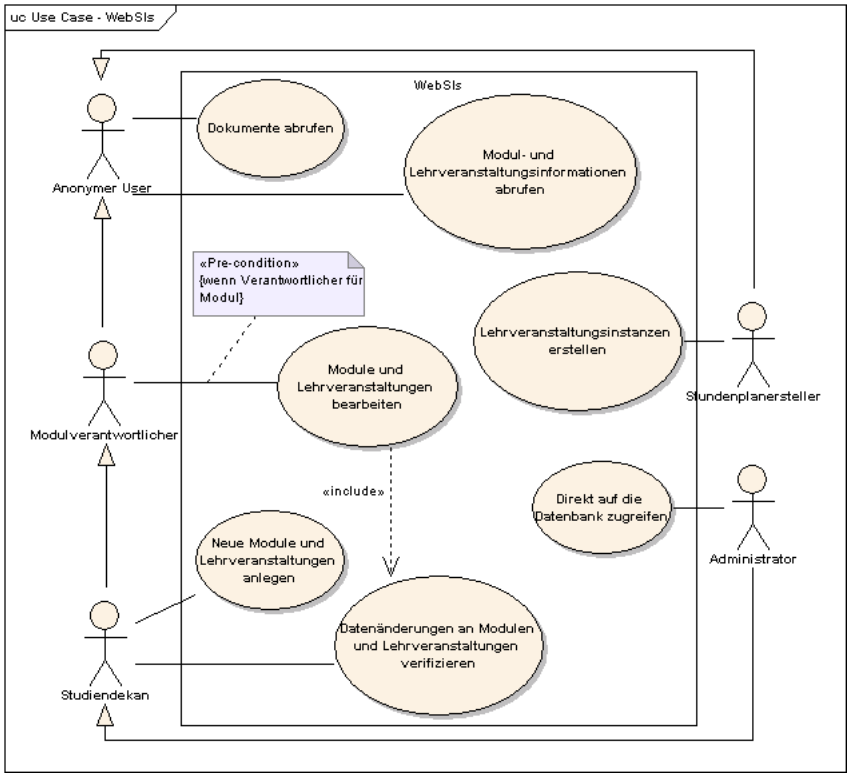

Abbildung 2.1: UseCase WebSis

Das zu entwickelnde System soll eine Reihe von Funktionen zur Verfügung stellen. Abbildung 2.1 zeigt diese zusammen mit den jeweiligen Benutzerrollen, welche in Abschnitt 2.3 genauer beschrieben werden.

So sollen aus den in einer Datenbank abgespeicherten Daten zu Modulen und Lehrveranstaltungen der Bachelor- und Masterstudiengängen dynamisch informative Webseiten generiert werden. Diese sind ohne Beschränkung und Authentifizierung für sämtliche Benutzergruppen öffentlich zugänglich.

Für die jeweiligen verantwortlichen Personen muss sowohl das Verändern von bestehenden Modul- und Lehrveranstaltungsdaten als auch das Anlegen von neuen Modulen und Lehrveranstaltungen über Web-Formulare ermöglicht werden. Ein Modulverantwortlicher (siehe Abschnitt 2.3) sollte bspw. die Daten seines angebotenen Moduls und den darin enthaltenen Lehrveranstaltungen editieren können. Diese Änderungen müssen jedoch immer vom Studiendekan (siehe Abschnitt 2.3) verifiziert werden, um zu jeder Zeit einen konsistenten Zustand der Lehrveranstaltungen zu haben.

Studierende und angehende Studenten sollen mehrere Dokumente bezüglich des Studiums abrufen können. So sollen sie etwa die Möglichkeit haben, durch die Auswahl eines Studiengangs und die Angabe von zusätzlichen Informationen - z.B. die Angabe des Nebenfachs beim Bachelorstudiengang Informatik oder der Vertiefung beim Masterstudiengang Informatik - ein entsprechendes Curriculum berechnen zu lassen. Dieses stellt eine Übersicht dar, welche die zu belegenden Module in den einzelnen Studiensemestern enthält. Die Regelungen der aktuellen Prüfungsordnung für die neuen Bachelor- und Masterstudiengänge (siehe [Uni07]) müssen hierfür berücksichtigt werden.

Für den Fachbereich 4 muss ein sog. Planungssheet berechnet und angezeigt werden. Dieser gibt an, wann welche Lehrveranstaltungen in der Vergangenheit angeboten wurden, welche zur Zeit stattfinden und welche in Zukunft angeboten werden. Des Weiteren lässt sich daraus erkennen, wann eine Veranstaltung mindestens wieder angeboten werden muss. Mithilfe dieser Informationen lassen sich die Instanzen von Vorlesungen für die kommenden Semester erzeugen. Diese werden vom Stundenplanersteller (siehe Abschnitt 2.3) verwaltet.

Es sollen weiterhin für die einzelne Module und Lehrveranstaltungen Kapazitätsdaten ausgewertet werden können. Diese stellen Schätzungen über die Anzahl der teilnehmenden Studierenden an einer Veranstaltung dar, wodurch bspw. die Raumplanung erleichtert werden soll.

2.2 Nicht-Funktionale Anforderungen

Zusätzlich zu den funktionalen Anforderungen des Systems gibt es noch eine Reihe weiterer Anforderungen an das System.

Aufgrund der Vielzahl anfallender Daten und deren Dynamik wird in WebSIs hoher Wert auf eine gut durchdachte Datenbank gelegt (siehe Abschnitt 3.1). Um die Korrektheit und Vollständigkeit dieser Daten garantieren zu können, muss das System entsprechende Plausibilitäts- und Konsistenzüberprüfungen durchführen

können.

Weiterhin soll WebSIs (in Zukunft) eine XML-Exportschnittstelle zur Verfügung stellen anhand welcher die Daten von WebSIs in das uniweite Content-Management-System der Universität Koblenz oder HISLSF[1] importiert werden können.

2.3 Benutzerverwaltung

Für die Administration von WebSIs werden verschiedene Benutzerrollen benötigt, welche im Folgenden beschrieben werden. Die Hierarchie der verschiedenen Benutzerrollen ist in Abbildung 2.1 dargestellt.

Anonymer User

Der anonyme User hat ausschließlich die Möglichkeit die Daten der Datenbank über das Webinterface abzufragen. Vertrauliche Informationen bekommt er nicht zu sehen. Weiterhin kann er sich einige generierte Daten anzeigen lassen.

Stundenplanersteller

Der Stundenplanersteller kann aus den Modulen und Lehrveranstaltungen Instanzen erzeugen. Er darf die dort enthalten Daten nicht ändern, jedoch ist er berechtigt für die Instanz relevante Daten (Vorlesungssemester, Vorlesungsjahr, URL zur Veranstaltung) hinzuzufügen. Eine Verifikation durch den Studiendekan ist nicht erforderlich.

Modulverantwortlicher

Der Modulverantwortliche hat über das Webinterface vollen Lesezugriff auf die Daten und darf zusätzlich die Module und die darin enthaltenen Lehrveranstaltungen editieren, für die dieser verantwortlich ist. Er darf jedoch nicht die Modulverantwortung abgeben, Module löschen oder erstellen. Seine Änderungen müssen durch den Studiendekan verifiziert werden.

[1] Web-Anwendung für *L*ehre, *S*tudium und *F*orschung (`http://www.his.de/abt1/ab10` (abgerufen am 27.06.2007))

Studiendekan

Der Studiendekan hat vollen Zugriff auf alle Module, Lehrveranstaltungen und sonstige Daten. Er muss über gemachte Änderungen des Modulverantwortlichen und des Administrators informiert werden und diese gegebenenfalls verifizieren.

Administrator

Der Administrator hat die selben Rechte wie der Studiendekan und hat zudem direkten Zugriff auf die Datenbank selbst. Seine Hauptaufgabe besteht in der Pflege und der Wartung der Datenbank.

3 Modellierung

3.1 Datenbankmodellierung

3.1.1 Beschreibung des konzeptuellen Datenmodells

Nachfolgend wird auf die Besonderheiten des konzeptuellen Modells eingegangen. Dieses wurde mit dem Modellierungswerkzeug „Enterprise Architect“[1] in der Version 6.5 erstellt, wobei sich weitestgehend an dessen Modellierungskonventionen für Klassendiagramme gehalten wurde.

Studiengänge

Die Klasse *CourseOfStudy* repräsentiert alle relevanten Bachelor- und Masterstudiengänge an der Universität Koblenz-Landau. Auf eine zunächst vorgenommene Trennung von Studienart (Master oder Bachelor) und der Fachrichtung (Informatik, Wirtschaftsinformatik, Computervisualistik und Informationsmanagement) wurde im finalen Entwurf verzichtet. Durch die Vereinigung beider in der Klasse *CourseOfStudy* wird eine Redundanz beim Studentenschwund im zweiten und im dritten Jahr in Kauf genommen. Diese Informationen entsprechen den Attributen *lossSecondYear* und *lossThirdYear*, welche ursprünglich nur Eigenschaften der Studienart waren. Da in Zukunft davon ausgegangen werden kann, dass hier auch eine Unterscheidung bei den einzelnen Studiengängen vorgenommen wird, erscheint die anfängliche Inkaufnahme der Redundanz als angemessen.

Zu jeder Studienart gehört eine Angabe zu der (Teilnehmer-)Größe in den einzelnen Veranstaltungstypen *(CourseType)*. Da die Studienart nicht als einzelne Klasse modelliert ist, sondern - wie bereits erwähnt - in die Klasse *CourseOfStudy* integriert wurde, wird auch hier eine gewisse Redundanz durch die Modellierung der Assoziation *hasCourseTypeSize* zwischen den Klassen *CourseOfStudy* und *CourseType* toleriert. Zudem könnte eine Unterscheidung der Teilnehmergröße nach dem Studiengang anstelle der Studienart wünschenswert sein.

[1] `http://www.sparxsystems.com.au/products/ea.html` (abgerufen am 27.06.2007)

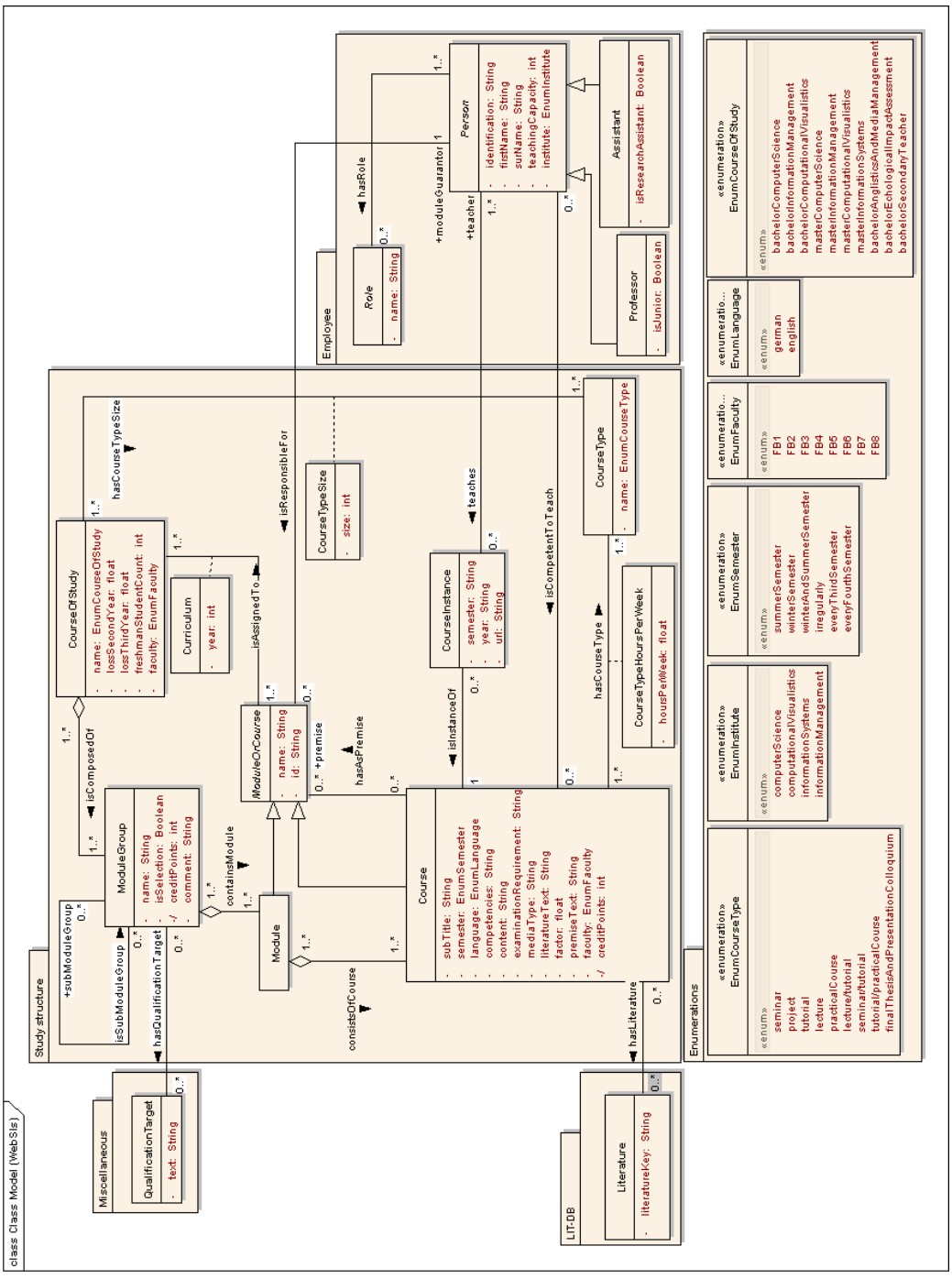

Abbildung 3.1: Konzeptuelles Datenbankmodell

Modulgruppen

Jeder Studiengang besteht aus beliebig vielen Modulgruppen *(ModuleGroup)*. Modulgruppen können selbst wiederum aus weiteren Submodulgruppen bestehen. Hiermit soll bspw. modelliert werden, dass eine Modulgruppe aus einem Pflicht- und einem Wahlpflichtbereich bestehen kann. Diese Situation wird dadurch gelöst, dass sowohl das Gesamtmodul als auch der Pflicht- und der Wahlpflichtbereich Modulgruppen sind. Letztere stellen dabei Submodulgruppen des Gesamtmoduls dar. Für jede Modulgruppe muss zudem zwischen Wahlpflicht und Pflicht (mit dem Attribut *isSelection*) unterschieden werden. Handelt es sich bei einer Modulgruppe um eine Pflichtgruppe, errechnen sich die Gesamtleistungspunkte aus der Summe aller in der Modulgruppe (und den Submodulgruppen) enthaltenen Module, während es bei Wahlpflichtgruppen explizit notiert werden muss.

Module und Lehrveranstaltungen

Module *(Module)* und Lehrveranstaltungen *(Course)* sind abgeleitete Spezialisierungsklassen der abstrakten Oberklasse *ModuleOrCourse*. Dies ist einerseits sinnvoll, da *Module* und *Course* gemeinsame Attribute enthalten, aber andererseits auch um ausdrücken zu können, dass eine Lehrveranstaltung sowohl ein Modul als auch eine andere Lehrveranstaltung als Voraussetzung haben kann. Dies wird durch die Assoziation *hasAsPremise* zwischen *Course* und *ModuleOrCourse* modelliert.

Jedes Modul besitzt genau einen Modulverantwortlichen. Dies wird durch die Beziehung *isResponsibleFor* ausgedrückt. Diese impliziert, dass es auch für jede Lehrveranstaltung einen Verantwortlichen geben muss. Jedoch ist die Rolle des Lehrveranstaltungsverantwortlichen für das zu entwickelnde System nicht relevant, da ein Modulverantwortlicher automatisch auch für alle im Modul enthaltenen Lehrveranstaltungen verwantwortlich ist.

Zu jeder Lehrveranstaltung existieren sog. Instanzen *(CourseInstance)*, welche zum Einen das Veranstaltungsjahr und das -semester beinhalten und zum Anderen mit den zugehörigen Lehrveranstaltungslehrenden (durch *teaches*) assoziiert sind.

Personen

Bei den Personen *(Person)* wird zwischen Professoren/ Juniorprofessoren und Mitarbeitern unterschieden. Jede Person kann zudem über eine Rolle (Studiendekan,

Stundenplanersteller) verfügen. Auf deren genaue Modellierung wurde an dieser Stelle verzichtet. Eine ausführliche Beschreibung der im System festgelegten Rechtevergabe für die einzelnen Rollen erfolgt in Abschnitt 2.3.

Enumerations

Für Attribute, die nur eine festgelegte Auswahl an Werten annehmen können, wurden entsprechende Enumeration-Klassen definiert. Die Klasse *EnumCourseOfStudy* enthält bspw. eine Auflistung aller für das System an der Universität Koblenz-Landau relevanten Bachelor- und Masterstudiengänge. Diese Klassen können bei Bedarf erweitert oder angepasst werden.

3.1.2 Die Überführung in das logische Datenbankschema

Im Folgenden wird die Überführung des oben erläuterten konzeptuellen Modells in das logische Datenbankschema beschrieben. Letzteres stellt die direkte Grundlage für die Implementierung einer relationalen Datenbank dar. Erstellt wurde dieses ebenfalls mit dem Modellierungswerkzeug „Enterprise Architect“ in der Version 6.5, wobei sich weitestgehend an dessen Modellierungskonventionen für Datenbankmodelle orientiert wurde.

Konventionen bei der Bezeichnervergabe

Als weitere (eigene) Konvention gilt zudem, dass Bezeichner für Tabellen und Attribute durchgehend mit Kleinbuchstaben geschrieben werden, um eventuelle Probleme mit Bezeichnern in MySQL im Voraus zu vermeiden. Zur besseren Lesbarkeit werden Unterstriche (_) als Trennzeichen zwischen Wörtern verwendet, wobei jedoch die Bezeichner von Klassen - außer bei Assoziationsklassen - ohne Trennzeichen übernommen werden. Aus *CourseOfStudy* wird demnach *courseofstudy* im logischen Datenbank, während die Assoziation *hasQualificationTarget* aus dem konzeptuellen Modell der Tabelle
has_qualificationtarget entspricht.

Die Umsetzung der Klassen

Alle Klassen (außer den Enumeration- und Assoziationsklassen) des konzeptuellen Modells, welche nicht Teil einer Spezialisierungshierarchie sind, werden im logi-

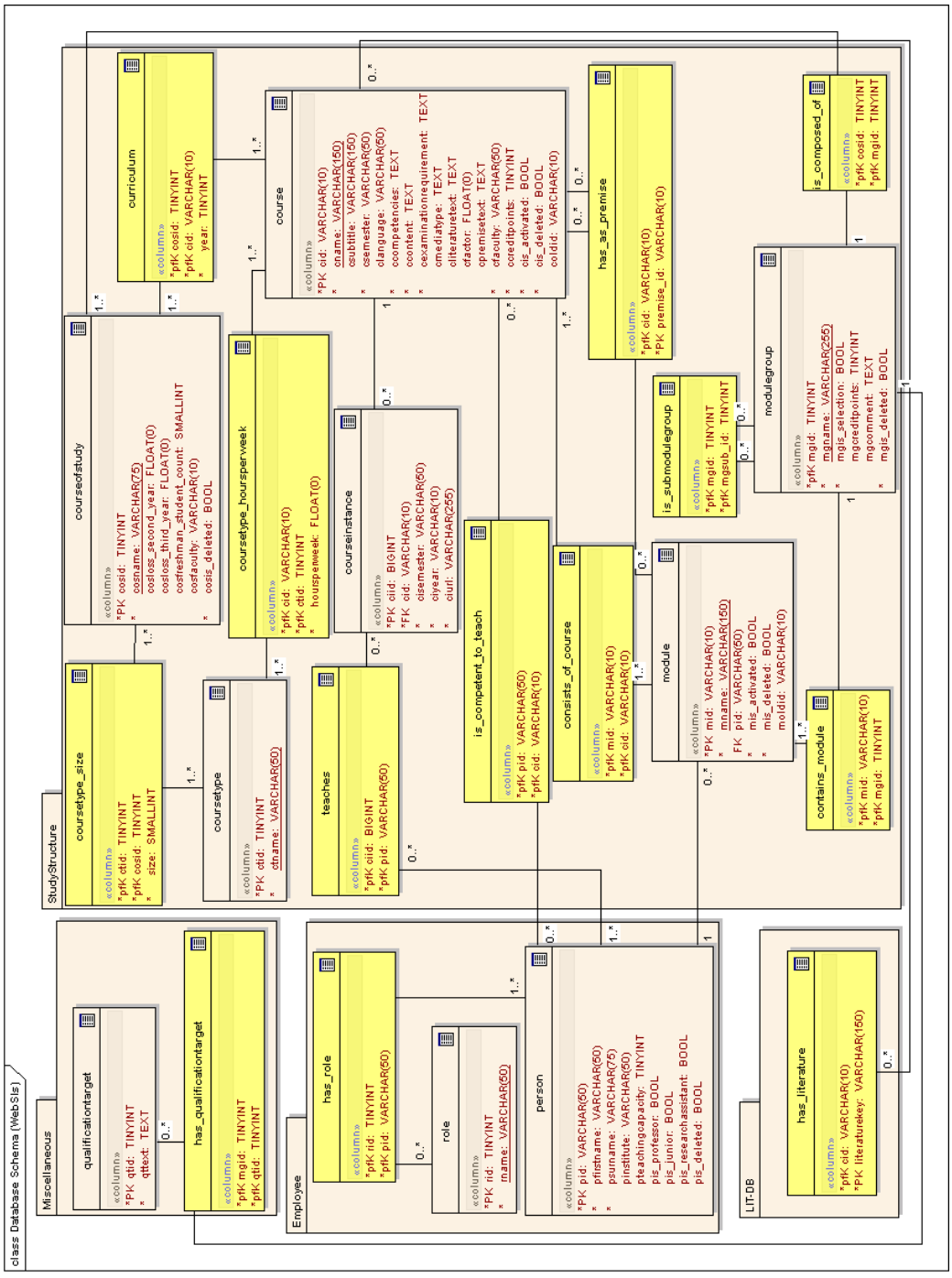

Abbildung 3.2: Logisches Datenbankschema

schen Datenbankmodell in eine Datenbanktabelle mit zugehörigem Namen transformiert. Die Attribute einer Klasse entsprechen dabei den Tabellenspalten. Der Spaltenbezeichner erhält ein dem Tabellennamen entsprechendes Kürzel als Präfix. Dies betrifft die Klassen *CourseOfStudy*, *ModuleGroup*, *Role*, *CourseInstance* und *QualificationTarget*. Die Klasse *Literature* im Package *LIT-DB* entspricht ebenfalls diesem Muster, wird jedoch nicht direkt umgesetzt, da sie repräsentativ für das uni-interne Literaturdatenbanksystem steht.

Die Umsetzung von Spezialisierungshierarchien

Bei allen Klassen, die entweder eine (abstrakte) Oberklasse oder eine von einer Oberklasse abgeleitete Subklasse sind, gibt es bei der Umsetzung in das logische Datenbankmodell verschiedene Möglichkeiten:

Zum Einen kann nur die (abstrakte) Oberklasse in eine Datenbanktabelle überführt werden, wobei diese die eigenen und alle Attribute sämtlicher abgeleiteter Subklassen als Tabellenspalten enthalten muss. Die Unterscheidung des Klassentyps muss dann zusätzlich über entsprechende (boolesche) Spalten durchgeführt werden.

Diese Variante wird für die abstrakte Oberklasse *Person* und deren abgeleiteten Subklassen *Professor* und *Assistant* im Package *Employee* angewendet. Im logischen Datenbankmodell existiert hierfür die Tabelle *person*, welche alle Attribute der Ober- und der einzelnen Subklassen enthält. Da nur zwischen zwei Subklassen (*Person* und *Assistant*) unterschieden werden muss, erfolgt deren Differenzierung in der Tabelle durch eine einzelne boolesche Spalte (*p*is_professor). Ist ihr Wert *true*, handelt es sich bei dem Eintrag um eine Instanz der Klasse *Professor*, ansonsten ist es eine Instanz der Klasse *Assistant*. Diese Vorgehensweise lohnt sich in diesem Fall aufgrund der geringen Attributanzahl in den Subklassen und aufgrund der Tatsache, dass nur zwei Klassentypen zu unterscheiden sind.

Wenn die Oberklasse abstrakt ist, können zum Anderen lediglich die abgeleiteten Subklassen transformiert werden. Die hier entstehenden Datenbanktabellen enthalten als Spalten alle geerbten Attribute der Oberklasse und die jeweils eigenen Klassenattribute. Diese Variante ist jedoch nur dann sinnvoll, wenn die Oberklasse als abstrakt definiert ist, so dass keine Instanzen der solchen im System auftreten können. Ansonsten müsste auch die Oberklasse in eine entsprechende Datenbanktabelle überführt werden.

Diese zweite Variante kommt bei der Oberklasse *ModuleOrCourse* und deren abgeleiteten Subklassen *Module* und *Course* im Package *StudyStructure* zum Einsatz. Im logischen Datenbankmodell entstehen die zu den jeweiligen Subklassen zugehörigen Datenbanktabellen *module* und *course*, welche jeweils zusätzlich zu den

eigenen Klassenattributen die Attribute der Oberklasse als Spalten enthalten. Das Attribut *id* der Oberklasse *ModuleOrCourse* wird dabei als eindeutiger Identifier für die Einträge in den Tabellen definiert. Dabei besteht dieser Identifier bei Modulen aus 4 Buchstaben und 2 nachfolgenden Ziffern (bspw. *INUF02*), während bei Lehrveranstaltungen ein Suffix - bestehend aus einem „-“ und einem Kleinbuchstaben (bspw. *IMGW03-a*) - hinzukommt. Dies gilt auch bei Modulen, die nur aus einer Lehrveranstaltung bestehen.

Für die Verwendung dieser Variante spricht einerseits die deutliche Differenz in der Attributanzahl der Klassen *Module* (2) und *Course* (14) und andererseits die Notwendigkeit im System eindeutig zwischen Modulen und Lehrveranstaltungen unterscheiden zu können.

Primärschlüssel

Alle bis hierhin erzeugten Tabellen erhalten eine zusätzliche (künstliche) Identifierspalte als Primärschlüssel um die Eindeutigkeit eines Datenbankeintrags zu gewährleisten, sofern sie kein Attribut besitzen, welches bereits die eindeutige Identifizierung einer Instanz ermöglicht. In letzterem Fall wird auf die Einführung eines künstlichen Primärschlüssels im logischen Datenbankmodell verzichtet. Dies trifft auf die Klassen *Module*, *Course* (jeweils durch das Attribut *id*, welches *cid* bzw. *mid* entspricht) und *Person* (durch das Attribut *identification*, welches *pid* entspricht) zu.

Die Umsetzung der Assoziationen

Assoziationen werden in der Regel ebenfalls in Datenbanktabellen mit gleichem Bezeichner überführt. Die tatsächliche Umsetzung der Assoziationen in das logische Datenbankmodell hängt dabei besonders von den definierten Kardinalitäten im konzeptuellen Modell ab. Hierfür lassen sich die folgenden Assoziationsarten aus relationaler Datenbanksicht unterscheiden:

- 1:1-Beziehung, d.h. jeder Instanz e1 der Klasse E1 ist höchstens eine Instanz e2 von der Klasse E2 zugeordnet;

- 1:N bzw. N:1-Beziehung, d.h. jeder Instanz e1 der Klasse E1 sind beliebig viele (auch keine) Instanzen e2 von der Klasse E2 zugeordnet, aber jede Instanz e2 der Klasse E2 steht mit maximal einer Instanz der Klasse E1 in Verbindung bzw. vice versa;

- N:M-Beziehung, d.h. dass jede Instanz e1 der Klasse E1 mit beliebig vielen (auch keinen) Instanzen e2 der Klasse E2 assoziiert werden kann und umgekehrt.

Im konzeptuellen Modell existieren keine 1:1-Beziehungen, sehr wohl aber zwei 1:N- bzw. N:1-Beziehungen. Die erste *isInstanceOf* zwischen den Klassen *CourseInstance* und *Course* ist (laut Leserichtung) eine N:1-Beziehung. Jede Kursinstanz ist demnach genau einem Kurs zugeordnet. Bei der Umsetzung in das logische Datenbankmodell bedeutet dies, dass die Tabelle *courseinstance* eine Spalte *cid* enthält, in welcher der entsprechende Primärschlüssel des zugehörigen Kurses eingetragen wird.

Die zweite Beziehung *isResponsibleFor* zwischen den Klassen *Person* und *ModuleOrCourse* stellt (laut Leserichtung) eine 1:N-Beziehung dar. Jedes Modul bzw. jede Lehrveranstaltung besitzt genau einen Verantwortlichen. Im logischen Datenbankmodell erhält die Tabelle *module* hierfür die Spalte *pid*, in welcher der Primärschlüssel der zugehörigen Person eingetragen wird. Diese Person ist somit der Modulverantwortliche für das referenzierte Modul. Laut dem konzeptuellen Modell müsste auch die Tabelle *course* eine entsprechende Spalte besitzen. Im System ist jedoch lediglich die Rolle des Modulverantwortlichen relevant. Die Rolle des Lehrveranstaltungsverantwortlichen entfällt und wird somit im logischen Datenbankmodell nicht berücksichtigt.

Bei der Assoziation *isAssignedTo* wurde analog vorgegangen: Im Datenbankmodell wird diese Beziehung auf die Klassen *CourseOfStudy* und *Course* beschränkt. Die Beziehung zwischen den Klassen *CourseOfStudy* und *Module* wird ignoriert, obwohl diese nach dem konzeptuellen Modell ebenfalls möglich wäre. Dieser Verzicht wird dadurch begründet, dass die Zuordnung eines Moduls zum Curriculum letztlich der Summe der Zuordnungen aller im Modul enthaltenen Lehrveranstaltungen entspricht und somit mit einer geeigneten SQL-Anfrage reproduzierbar ist.

Besitzt demnach eine Assoziation auf einer ihrer beiden Enden die Kardinalität *1* erhält diejenige Tabelle eine zusätzliche Spalte, deren zugehörige Klasse die Kardinalität N besitzt. In diese wird der Primärschlüssel der Instanz der Klasse gespeichert, die mit der Kardinalität *1* versehen ist.

Bei N:M-Beziehung werden aus den Assoziationen des konzeptuellen Modells entsprechende Datenbanktabellen im logischen Modell erzeugt. Diese enthalten die jeweiligen Primärschlüssel aller Datenbanktabellen, welche sie verbinden. Die Menge aller referenzierten Primärschlüssel bildet dabei den Primärschlüssel der Assoziationstabelle. Dies betrifft die Assoziationen *isComposedOf*, *isSubModuleGroup*, *containsModule*, *consistsOfCourse*, *isAssignedTo*, *hasAsPremise*, *teaches*, *isCom-*

petentToTeach, *hasCourseType*, *hasRole*, *hasLiterature* und *hasQualificationTarget*.

Koventionen bei der Umsetzung von Modulgruppen

Durch den Verzicht auf eine Verschmelzung der Assoziationen *isComposedOf* und *containsModule* zu einer ternären Beziehung zwischen den Klassen *CourseOfStudy*, *ModuleGroup* und *Module* - welche in diesem Fall zu einer Transformation in eine einzelne Datenbanktabelle geführt hätte - müssen bestimmte Konventionen bei der Erstellung und der Bezeichnung von Modulgruppen beachtet werden. Da stattdessen zwei getrennte Datenbanktabellen für die Assoziationen *isComposedOf* und *containsModule* existieren, ist es nicht möglich abzuspeichern, dass eine Modulgruppe in verschiedenen Studiengängen unterschiedliche Module enthalten kann. Dies wäre mit der oben erwähnten ternären Beziehung möglich gewesen, wobei dann auch zusätzlich eine Unterscheidung bei der Assoziation *isSubModuleGroup* nach Studiengang notwendig wäre. Kommt daher eine Modulgruppe in mehreren Studiengängen mit jeweils unterschiedlichem Aufbau vor, wird diese in der Tabelle *modulegroup* für jeden Studiengang jeweils mit der Bezeichnung „*<Modulgruppe> (<Studiengang>)* separat gespeichert. Jede Modulgruppe besteht damit immer aus der gleichen Menge von Modulen.

Die Umsetzung von Assoziationsklassen

Enthält eine Assoziation im konzeptuellen Modell eine Assoziationsklasse (siehe *Curriculum*, *CourseTypeSize* und *CourseTypeHoursPerWeek*) wird diese anstelle der Assoziation als Datenbanktabelle mit zugehörigen Attributen umgesetzt. Datenbanktabellen, die aus einer Assoziation oder einer Assoziationsklasse des konzeptuellen Modells transformiert wurden, sind im logischen Datenbankmodell gelb markiert.

Benutzerverwaltung

Die Benutzerverwaltung, welche im konzeptuellen Modell durch die Assoziation *hasRole* und die Klasse *Role* angedeutet wurden, wird im logischen Modell in den entsprechenden Tabellen *(has_role bzw. role)* abgewickelt. Hier werden jedoch lediglich die Rollen des Studiendekans und des Stundenplanerstellers abgespeichert und mit den zugehörigen Personeninstanzen assoziiert; die restlichen im System definierten Rollen (siehe Abschnitt 2.3) werden über andere Relationen ausgedrückt.

Enumerations

Auf die Umsetzung der Enumeration-Klassen im Package *Enumerations* in der Datenbank wird verzichtet. Implementiert werden diese stattdessen in der Anwendungslogik. Deren Werte werden als simple Zeichenketten in den jeweiligen Tabellenspalten gespeichert.

3.1.3 Weitere Bemerkungen

Änderungen an Modulen und Lehrveranstaltungen

Bei den Tabellen *course* und *module* wurden jeweils eine Spalte *cis_activated* bzw. *mis_activated* eingeführt, welche einen Boolean-Wert enthalten.

Wenn ein Modulverantwortlicher Änderungen an einem Modul bzw. einer Lehrveranstaltung durchführt, wird ein zusätzlicher Datensatz angelegt, welcher die geplanten Änderungen speichert. Um dabei die Eindeutigkeit der Kürzel in den Tabellen zu gewährleisten, erhalten die Kürzel der Datensätze, welche die Änderungen enthalten, das Suffix „TP“. Der Wert der *is_activated*-Spalte für diesen Datensatz wird auf *false* gesetzt. Dies soll zum Ausdruck bringen, dass die Modul- bzw. die Lehrveranstaltungsänderung bei einer SQL-SELECT-Query nicht in die Ergebnismenge aufgenommen wird, solange sie nicht vom Studiendekan verifiziert worden ist. Demnach wird so lange die „veraltete“ Version eines Moduls bzw. einer Lehrveranstaltung angezeigt, bis der Studiendekan über die Gültigkeit der gemachten Änderungen entschieden hat.

Um die Zugehörigkeit eines Änderungseintrags zu dem Originaleintrag reproduzierbar und eindeutig zu machen, erhalten die Tabellen *course* und *module* zusätzlich die Spalten *coldid* bzw. *moldid*. In diese wird bei einer Lehrveranstaltungs- bzw. Moduländerung das Kürzel der zu editierenden Lehrveranstaltung bzw. des zu editierenden Moduls eingetragen. Wird also z.B. das Modul *INJE01* editiert und dessen Kürzel auf *INMO01* verändert, wird als Kürzel für den Editiereintrag *INMO01TP* in der Datenbank eingetragen. Dieser neue Eintrag erhält für seine Spalte *moldid* als Wert das Kürzel des editierten Moduls, sprich *INJE01*.

Entscheidet sich der Studiendekan für die Übernahme dieser - für den Studiendekan selbst zusätzlich noch editierbaren - Änderungen, werden die geänderten Daten im entsprechenden Datensatz für das editierte Modul bzw. die editierte Lehrveranstaltung übernommen. Wie bereits im vorigen Abschnitt erwähnt, lässt sich durch den Wert der Spalte *moldid* bzw. *coldid* des Editiereintrags das betroffene Modul bzw. die betroffene Lehrveranstaltung feststellen. Der Editiereintrag

selbst wird aus der zugehörigen Tabelle gelöscht, da die Daten des betroffenen Moduls bzw. der betroffenen Lehrveranstaltung durch die erfolgreiche Verifikation bereits erfolgreich aktualisiert worden sind.

Lehnt der Studiendekan die an einer Lehrveranstaltung bzw. einem Modul durchgeführten Änderungen ab, muss lediglich der Datensatz aus der Datenbank gelöscht werden, welcher die Änderungen speichert.

Löschen von Studiengängen, Modulgruppen, Modulen, Lehrveranstaltungen und Personen

Für die Tabellen *course*, *coursesofstudy*, *module*, *modulegroup* und *person* wurden jeweils eine Spalte *cis_deleted*, *cosid_deleted*, *mis_deleted*, *mgis_deleted* bzw. *pis_deleted* eingeführt, die einen Boolean-Wert enthalten. Wenn der Studiendekan entscheidet, eine Lehrveranstaltung, einen Studiengang, eine Modulgruppe oder eine Person zu löschen, werden deren Informationen nicht direkt aus der Datenbank entfernt. Stattdessen wird der Boolean-Wert in der entsprechenden *is_deleted*-Spalte auf *true* gesetzt. Endgültig gelöscht wird ein Datensatz, welcher den Wert *true* in der *is_deleted*-Spalte enthält also erst dann, wenn eine weitere Löschoperation durch den Studiendekan angestoßen wird. Zudem lassen sich Datensätze, deren Wert in der *is_deleted*-Spalte auf *true* gesetzt sind, durch den Studiendekan reaktivieren. Nach der Reaktivierung wird der Wert in der *is_deleted*-Spalte entsprechend auf *false* zurückgesetzt. Durch diesen Mechanismus wird ein „Papierkorb“ simuliert.

Datensätze in den Tabellen *course*, *courseofstudy*, *module*, *modulegroup* und *person*, deren *is_deleted*-Spalte den Wert *true* haben, lassen sich weder editieren noch werden diese bei einer SQL-SELECT-QUERY in die Ergebnismenge aufgenommen.

3.2 Modellierung der Anwendungslogik

Die Modellierung der Anwendungslogik orientiert sich an der eines Wikis. Die Interaktion mit dem Benutzer geschieht immer über eine zentrale `index.php5`-Datei. Anhand des GET-Parameters *action* wird dann ermittelt, welche Aktion durchzuführen ist. Dementsprechend wird in dem jeweiligen Unterordner eine weitere, lokale `index.php5`-Datei aufgerufen. Diese steuert nun den weiteren Ablauf der Aktion, welcher in Abbildung 3.3 beschrieben ist.

Wenn es sich um eine Member-Aktion (Aktionen, welche einen Login erfordern)

handelt wird nach dem Login meist zuerst ein Handler aufgerufen, welcher dazu dient die Daten aus den, meist in den Member-Aktionen vorhandenen, Formularen zu verarbeiten. Danach wird eine Visualisierungsklasse aufgerufen. Diese dient dem Generieren der Bildschirmausgabe der Aktion. Diese Visualisierungsklassen bekommen ein Datenobjekt übergeben, welches dafür alle nötigen Informationen enthält. Diese wurden entweder vom Handler verarbeitet oder aus der MySQL-Datenbank ausgelesen. Für die Verbindung mit der Datenbank ist eine Datenbankklasse zuständig. Diese verarbeitet alle Vorgänge, welche eine Kommunikation mit der Datenbank erfordern.

Wurde das Formular bei einer Member-Aktion abgeschickt, so wird wieder die jeweilige Aktion aufgerufen und der Handler beginnt mit der Verarbeitung der Formulardaten. Wenn die Verarbeitung gelingt, so werden die erstellten Daten angezeigt, ansonsten wird das Formular mit den vorher eingegebenen Daten und den jeweiligen Fehlermeldungen erneut ausgegeben.

Bei dem gesamten Ablauf nutzen alle Klassen, mit Ausnahme der Datenbankklasse, immer wieder Utilityklassen, welche Hilfsfunktionen (z.B. für das Generieren von Überschriften in der HTML-Ausgabe) enthalten.

3.2.1 Datenklassen

Die Datenklassen dienen dem Speichern der Daten für die weitere Verarbeitung. Sie bieten außerdem Funktionen zum Speichern, Löschen und Aktualisieren der Daten in der Datenbank.

Abbildung 3.4 zeigt die Klasse `CourseOfStudy`, welche die Informationen für einen Studiengang bereitstellt. Da bei PHP das Überladen von mehreren Konstruktoren nicht möglich ist, aber ein Konstruktor zum Auslesen der Daten aus der Datenbank, sowie ein weiterer zum Speichern neuer Daten aus den Formularen benötigt wird, musste dort eine etwas ungewöhnliche Modellierung erfolgen: Der Parameter `$useDataBase` entscheidet in diesem Fall darüber, ob die Daten aus der Datenbank gelesen werden oder nicht. Sollen die Daten aus der Datenbank extrahiert werden, so wird lediglich der Parameter `$id` benötigt. Alle weiteren Parameter werden nicht berücksichtigt. Im Falle, dass die Daten manuell gesetzt werden, wird die Variable `$isCreatedManually` auf `true` gesetzt. Dies bewirkt, das bei einigen Get-Methoden, welche die Daten immer direkt aus der Datenbank extrahieren, stattdessen nur der Wert der jeweiligen internen Variable zurückgegeben wird.

Beim Speichern der Daten in der Datenbank können Fehler auftreten, welche die gesamte Transaktion abbrechen und eine Benutzerinteraktion erfordern. Für die-

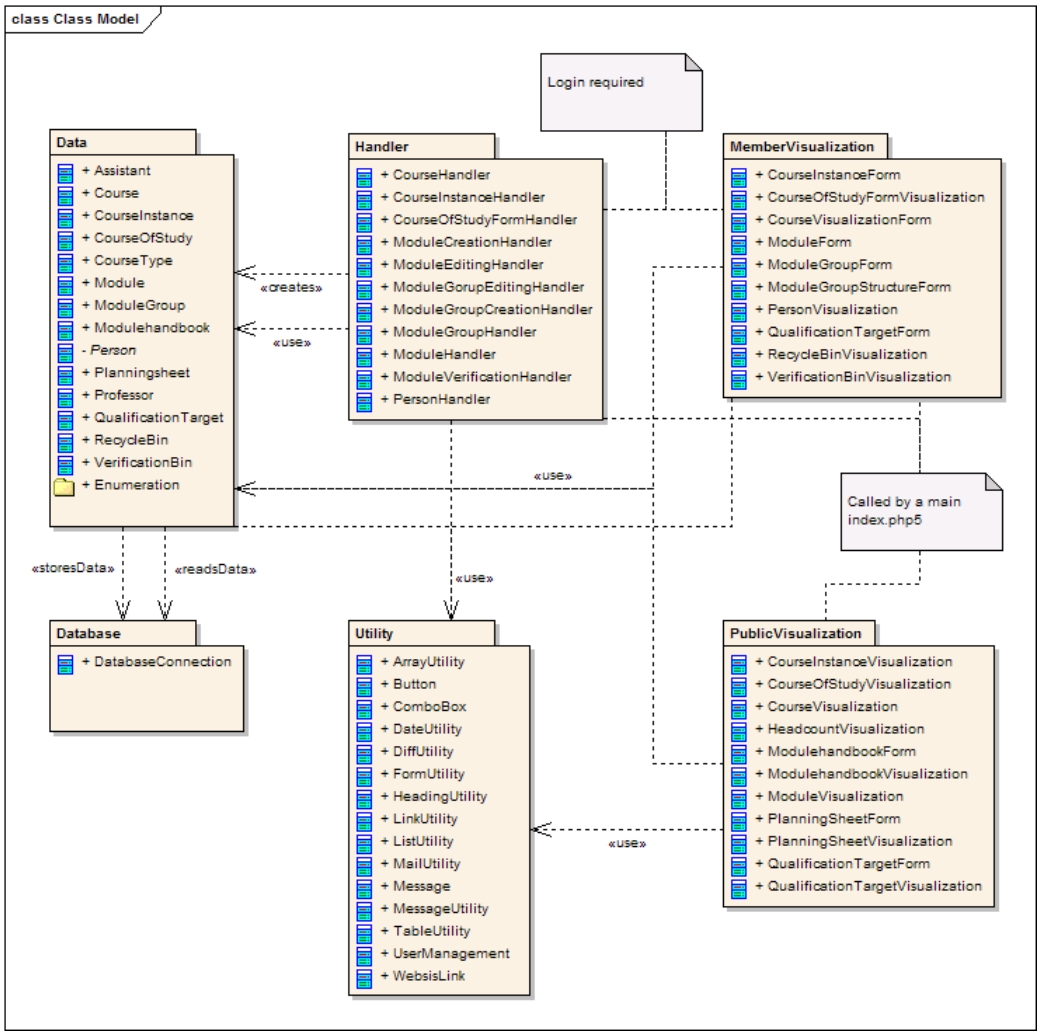

Abbildung 3.3: Aufbau der Anwendungslogik

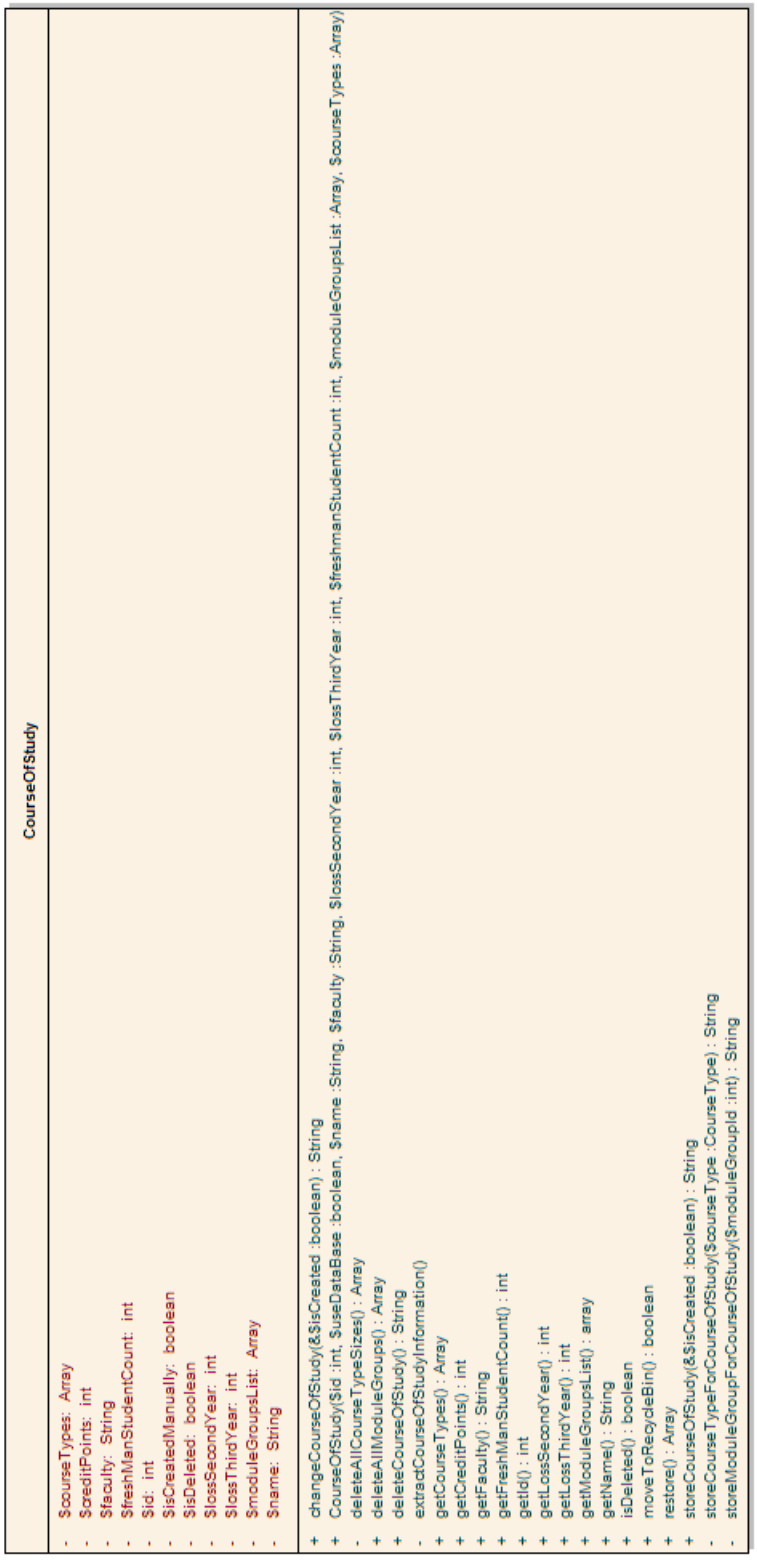

Abbildung 3.4: Aufbau der Datenklasse `CourseOfStudy`

sen Fall wird den Funktionen, welche in die Datenbank schreiben, die Variable `$isCreated` per *Call By Reference* übergeben. Dies dient der Unterscheidung zwischen dem Editieren und dem Erstellen von Einträgen in der Datenbank. Während bei einem korrekt erstellten Eintrag in der Datenbank, die Visualisierung der gespeicherten Daten aufgerufen wird, wird bei einem Fehlschlag nochmals das Formular für das Erstellen der Daten angezeigt.

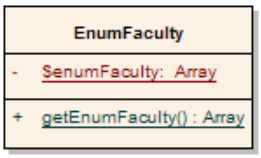

Abbildung 3.5: Aufbau der Enumerationklasse `EnumFaculty`

Zu den Datenklassen gehören auch die Enumerationklassen (siehe Abbildung 3.5). Diese beinhalten alle Werte, welche der dementsprechenden Variable zugeordnet werden dürfen (z.B. `$faculty` darf nur Werte aus dem Array `$enumFaculty` enthalten). Eine solche Klasse besitzt ein Array, welches alle möglichen Werte beinhaltet, und eine Getter-Methode um diesen Array auszulesen. Hauptsächlich finden diese Klassen beim Befüllen von ComboBoxen Verwendung.

3.2.2 Handler

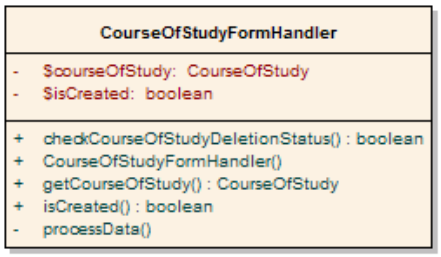

Abbildung 3.6: Aufbau der Handlerklasse `CourseOfStudyFormHandler`

Die Handlerklassen verarbeiten die Daten, welche in die Formulare eingegeben werden. Dabei überprüfen sie zusätzlich die Eingabe auf Fehler. Wenn beispielsweise die Variable `$lossSecondYear` aus Abbildung 3.4 einen Wert kleiner 0 oder größer 1 einnimmt, dann wird die bereits in Unterabschnitt 3.2.1 erwähnte Variable `$isCreated` auf `false` gesetzt und am Ende der Verarbeitung wird wieder das Formular mit der Fehlerausgabe angezeigt.

Sollte alles fehlerfrei abgelaufen sein, so wird ein Datenobjekt der jeweiligen Klasse erstellt und die Daten, wenn dies gewünscht wurde, auch in die Datenbank

geschrieben. Dies wird durch einen Aufruf der jeweiligen Funktionen im Datenobjekt bewirkt.

3.2.3 Datenbankklasse

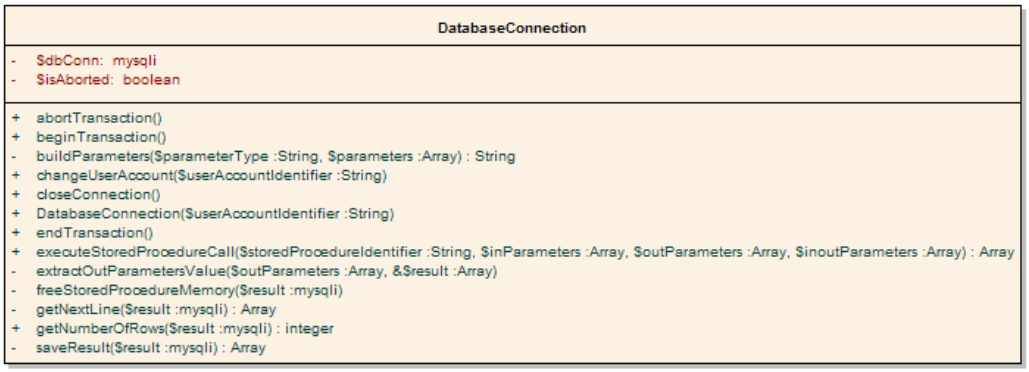

Abbildung 3.7: Aufbau der Datenbankklasse `DatabaseConnection`

Die Datenbankklasse `DatabaseConnection` (siehe Abbildung 3.7) steuert die Kommunikation mit der Datenbank. Auf die genauere Funktionsweise dieser Klasse wird in Unterabschnitt 4.2.2 eingegangen.

3.2.4 Visualisierungsklassen

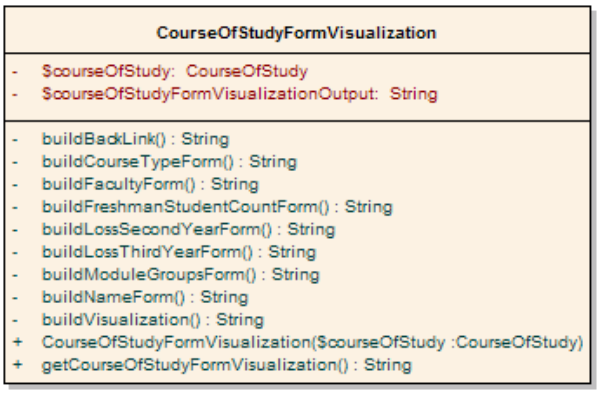

Abbildung 3.8: Aufbau der Member-Visualisierungsklasse `CourseOfStudyFormVisualization`

Die Visualisierungsklassen generieren die HTML Ausgabe. Die Member-Visualisierungsklassen (siehe Abbildung 3.8) greifen dabei auf das vom Handler erstellte Datenob-

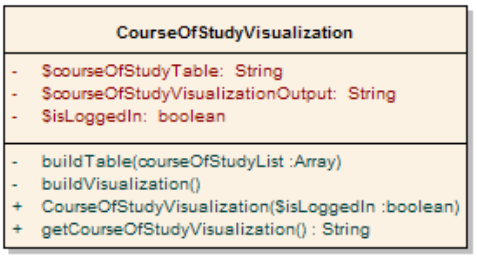

Abbildung 3.9: Aufbau der öffentlichen Visualisierungsklasse `CourseOfStudy-Visualization`

jekt zu, die öffentlichen Visualisierungsklassen (siehe Abbildung 3.9) extrahieren die Daten direkt aus der Datenbank.

Auffällig ist bei der *öffentlichen* Visualisierungsklasse der Konstruktorparameter `$isLoggedIn`. Dieser ist notwendig, weil die Studiengangsinformationen ebenfalls im Memberbereich angezeigt werden sollen. Dort sollen aber weitere Zusatzinformationen dargestellt werden, welche nicht für die Öffentlichkeit zugänglich sein sollen. Durch das Prüfen dieser Variable werden diese Informationen dargestellt oder nicht.

3.2.5 Utilityklassen

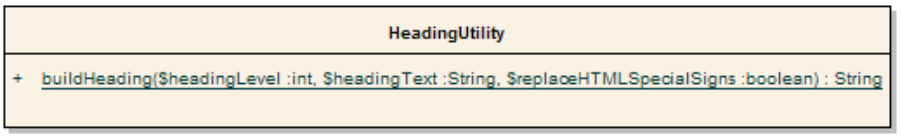

Abbildung 3.10: Aufbau der Utilityklasse `HeadingUtility`

Die Utilityklassen enthalten kleine Werkzeuge, die immer wieder Verwendung finden. So enthält die Klasse `HeadingUtility` aus Abbildung 3.10 beispielsweise die Funktion `buildHeading()`. Diese erstellt eine Überschrift in HTML, welche dem Design der Universität Koblenz entspricht.

4 Implementierung

4.1 Datenbankimplementierung

4.1.1 Implementierung des logischen Datenbankmodells

Das Datenbankmanagementsystem

Als Datenbankmanagementsystem (DBMS) wird MySQL in der Version 5.0.26 verwendet. Dieses läuft auf einem Server des Rechenzentrums und wird dementsprechend gewartet und gepflegt, so dass beim Eintritt von Problemen (beispielsweise Datenverlust) auf kompetenten Support und im Notfall auf entsprechende vom Rechenzentrum erstellte Datenbackups zurückgegriffen werden können. Zu erreichen ist das DBMS unter der Adresse *mysqlhost.uni-koblenz.de*.

Der Verzicht der Inbetriebnahme eines separaten DBMS auf einem Server außerhalb des Rechenzentrums bringt zwar zum Einen u.a. den Vorteil der erwähnten Recovery-Kompenente, hat aber zum Anderen den Nachteil, dass man die Konfigurationseinstellungen des DBMS im Rechenzentrum berücksichtigen muss, wodurch einige Einschränkungen bei der Datenbankimplementierung in Kauf genommen werden müssen. Weitere Restriktionen enstehen zudem dadurch, dass die im Rechenzentrum verwendete MySQL-Version nicht der zur Zeit aktuellsten Version (5.1.21-beta) entspricht. So muss aus diesem Grund auf die Verwendung von Triggern auf der Datenbank verzichtet werden.

Die Einrichtung der Datenbank

An der Universität Koblenz-Landau können neue Datenbanken über ein Webinterface[1] angelegt werden. Hierbei muss man sich mit seiner Unikennung anmelden. Nach erfolgreicher Anmeldung kann ein *Datenbankname* und ein zugehöriges *Datenbankpasswort* eingegeben werden.

[1] `http://www.uni-koblenz.de/GHRKO/MysqlDatenbank` (abgerufen am 10.07.2007)

Zudem hat man als Eigentümer der Datenbank die Möglichkeit über ein weiteres Webinterface neue Benutzer auf der Datenbank einzurichten und diesen eine limitierte Anzahl von Rechten zu übertragen. Dies ist ausschließlich über das erwähnte Webinterface möglich, da man an dieser Stelle den Adminaccount verwendet, welcher als einziger über die notwendigen Berechtigungen zum Anlegen von Benutzern verfügt. Näheres zur Benutzerkontoverwaltung und zur Rechtevergabe findet sich in Unterabschnitt 4.1.2.

Die offizielle Datenbank wird unter dem Account „websis“ eingerichtet.

Die Datenbankadministration

Zur Unterstützung der Datenbankadministration stellt das Rechenzentrum das Tool phpMyAdmin in der Version 2.8.0.2[2] zur Verfügung. Hierüber kann sich der Datenbankeigentümer einloggen, um neue Tabellen zu erzeugen und bestehende Tabellen zu löschen, SQL-SELECT-Querys durchzuführen und Datensätze einfügen, aktualisieren und löschen zu können.

Das Erstellen von neuen Benutzerkonten ist über phpMyAdmin nicht möglich, da der Datenbankeigentümer nicht über ausreichende Berechtigungen verfügt, um solche Aktionen durchzuführen. Neue Benutzerkonten mit entsprechender Rechtevergabe müssen - wie bereits oben erwähnt - über das entsprechende Webinterface eingericht werden.

Die Implementierung der Datenbanktabellen

Das zur Erstellung des logischen Datenbankmodells verwendete Modellierungswerkzeug Enterprise Architect bietet die Möglichkeit an, unter der Rubrik *Code Engineering* aus einem bestehenden logischen Datenbankmodell die zugehörigen SQL-Data Definition Language-Befehle (SQL-DDL-Befehle) automatisiert erzeugen zu lassen. Es werden also die notwendigen SQL-Befehle zur Implementierung der im Modell definierten Datenbanktabellen generiert. Dabei kann man sich auf Wunsch auch zusätzlich die notwendigen Befehle zur Erstellung u.a. von Primär- und Fremdschlüsseln, Indizes etc. erzeugen lassen. Enterprise Architect unterstützt dabei auch das Reverse Engineering, d.h. die Erzeugung eines logischen Datenbankmodells aus einer bestehenden Datenbankimplementierung. Dieses Feature wurde im Rahmen der Studienarbeit jedoch nicht benötigt.

[2] `http://mysqlhost.uni-koblenz.de/phpMyAdmin/index.php` (abgerufen am 10.07.2007)

Die automatisiert generierten SQL-Befehle waren syntaktisch korrekt und ließen sich problemlos über das SQL-Anfragefenster von phpMyAdmin ausführen. Lediglich einige wenige überflüssige Indizes mussten manuell aus den entsprechenden Datenbanktabellen entfernt werden.

Die InnoDB Tabellen

In MySQL sind diverse Tabellentypen bzw. Speicherengines definiert[3]. Der Standardtyp *MyISAM* arbeitet effektiv bei der Datenspeicherung und bietet u.a. Volltext-Suchfähigkeiten, unterstützt jedoch keine Transaktionen und ist somit nicht transaktionssicher. Der im Kontext der Studienarbeit verwendete Tabellentyp ist daher *InnoDB*. InnoDB unterstützt das sog. *ACID-Paradigma*.

Im Folgenden werden die einzelnen Bestandteile des ACID-Paradigmas kurz erläutert[KE04]:

Atomicity — (Atomarität) besagt, dass eine Transaktion als die kleinste nicht weiter zerlegbare Einheit behandelt werden muss. Bei einer Transaktionsdurchführung werden entweder alle Änderungen in der Datenbank übernommen oder gar keine. Hier wird das sog. „alles-oder-nichts“-Prinzip verwendet.

Consistency — (Konsistenz) bedeutet, dass nach Beendigung einer Transaktion ein konsistenter Zustand der Daten vorliegen muss. Dabei sind inkonsistente Zwischenzustände erlaubt, jedoch muss der Endzustand die im Datenbankschema definierten Konsistenzbedingungen einhalten.

Isolation — bezeichnet die Nebenläufigkeit von mehreren parallel ausgeführten Transaktionen, die sich jedoch nicht gegenseitig beeinflussen und somit fehlerhafte Endzustände und Inkonsistenzen vermeiden.

Durability — (Dauerhaftigkeit) besagt, dass eine erfolgreich durchgeführte Transaktion dauerhaft - auch nach einem Hardware- oder Softwarefehler - in der Datenbank erhalten bleiben muss. Im Notfall muss also eine vollständige Datenwiederherstellung gewährleistet sein.

Laut den MySQL-Entwicklern ist „InnoDB eine transaktionssichere (ACID-konforme) Speicher-Engine mit Commit-, Rollback- und Datenwiederherstellungsfähigkeiten“ [MyS07].

[3] `http://dev.mysql.com/doc/refman/5.1/de/storage-engines.html` (abgerufen am 10.07.2007)

Zudem unterstützt InnoDB - im Gegensatz zu *MyISAM* - die Verwendung von Fremdschlüsseln, die ebenfalls im logischen Datenbankmodell definiert worden sind. Näheres hierzu findet sich in Abschnitt 4.1.1.

Auf dem DBMS des Rechenzentrums der Universität Koblenz-Landau ist bei der Erzeugung neuer Datenbanktabellen die InnoDB-Engine als Standard definiert. Wäre InnoDB jedoch nicht als Standard definiert, kann durch die zusätzliche Angabe von „ENGINE = InnoDB„ bei dem *CREATE TABLE*-Befehl sichergestellt werden, dass eine InnoDB-Tabelle angelegt wird. Voraussetzung ist dabei, dass die InnoDB-Speicherengine auf dem DBMS aktiviert wurde. Der SQL-Befehl zur Erzeugung der Tabelle module würde somit wie in Auflistung 4.1 definiert aussehen.

```
1  CREATE TABLE module
2  (
3    mid VARCHAR(10) NOT NULL,
4    mname VARCHAR(150) NOT NULL,
5    pid VARCHAR(50) NOT NULL,
6    mis_activated BOOL NOT NULL,
7    moldid VARCHAR(10),
8    PRIMARY KEY (mid),
9    INDEX INDEX_module (mname ASC)
10 )
11 ENGINE = InnoDB;
```

Auflistung 4.1: SQL-DDL für die Tabelle module

Die Tabellen- und Spaltenbezeicher bei InnoDB-Tabellen sollten durchgehend Kleinbuchstaben enthalten, da beispielsweise das Betriebssystem Windows die Datenbank-, Tabellen- und Spaltennamen immer in Kleinbuchstaben speichert.

Die Fremdschlüssel

Fremdschlüssel (Foreign Keys) bezeichnen Tabellenspalten einer Datenbanktabelle (Kindtabelle), die auf Primärschlüsselspalten einer anderen Tabelle (Elterntabelle) referenzieren. Dies bedeutet u.a., dass in der Kindtabelle in den Fremdschlüsselspalten nur Werte eingetragen werden können, welche in der Elterntabelle in den referenzierten Primärschlüsselspalten bereits gespeichert wurden.

InnoDB unterstützt als einziger Tabellentyp in MySQL die sog. FOREIGN KEY-Constraints bzw. die Definition von Fremdschlüsseln. InnoDB-Tabellen weisen dabei jede INSERT- bzw. UPDATE-Operation zurück, die versucht einen Fremschlüsselwert in einer Kindtabelle anzulegen, wenn kein passender Schlüsselwert in der Elterntabelle gefunden wird [MyS07].

Im logischen Datenbankmodell sind die entsprechenden Tabellenspalten, welche einen Fremdschlüssel bilden, mit dem Kürzel *FK* versehen.

Entscheidend bei der Definition von Fremdschlüsseln ist v.a. wie die Datenbank reagieren soll, wenn in der Elterntabelle der Wert einer durch einen Fremdschlüssel referenzierten Spalte verändert (ON UPDATE) oder gelöscht wird (ON DELETE) und was mit den entsprechenden Einträgen in der Kindtabelle passieren soll.

Für diese Fälle bietet MySQL u.a. folgende Optionen an [MyS07]:

CASCADE — Wird eine Zeile in der Elterntabelle gelöscht oder geändert, werden die zugehörigen Zeilen in der Kindtabelle automatisch gelöscht bzw. entsprechend geändert. Hier sind also *ON DELETE CASCADE* und *ON UPDATE CASCADE* möglich.

SET NULL — Wird eine Zeile in der Elterntabelle gelöscht oder geändert, werden die Werte in zugehörigen Fremdschlüsselspalten der Kindtabelle auf NULL gesetzt, sofern diese Spalten nicht als NOT NULL definiert worden sind. Hier sind demnach die Optionen *ON DELETE SET NULL* und *ON UPDATE SET NULL* definierbar.

NO ACTION — Verhindert dass der Wert einer referenzierten Primärschlüsselspalte geändert oder gelöscht werden kann, solange dieser in einem Fremdschlüssel einer Kindtabelle vorkommt. „InnoDB weist dann die Lösch- oder Änderungsoperation auf der Elterntabelle zurück.“

RESTRICT — Kann semantisch äquivalent zu NO ACTION verwendet werden. Lässt man die ON DELETE- oder die ON UPDATE-Klauseln bei der Definition von Fremdschlüsseln weg, entspricht dies der Verwendung von RESTRICT bzw. NO ACTION.

Bei der Implementierung der Fremdschlüssel wurde in der Tabelle *module* die Einstellung *ON DELETE SET NULL* und *ON UPDATE CASCADE* verwendet, da ein Eintrag in dieser Tabelle nicht gelöscht werden soll, wenn ein zu der Spalte *pid* zugehöriger Eintrag in der Tabelle *person* entfernt wird. Ansonsten wurden durchgehend die Klauseln *ON UPDATE CASCADE* und *ON DELETE CASCADE* verwendet, so dass alle Lösch- und Änderungsoperationen auf referenzierten Primärschlüsselspalten in der Elterntabelle in den entsprechenden Fremdschlüsselspalten der Kindtabelle übernommen werden.

Die automatische Erzeugung der SQL-DDL-Befehle durch den Enterprise Architect trennt die Definition der Fremdschlüssel von der Definition der zugehörigen Tabellen.

Die Fremdschlüsseldefinition für die Tabelle *contains_module* stellt sich demnach wie in Auflistung 4.2 definiert dar.

```
1 ALTER TABLE contains_module ADD CONSTRAINT FK_mgid_contains_module
2 FOREIGN KEY (mgid) REFERENCES modulegroup (mgid)
3 ON DELETE CASCADE ON UPDATE CASCADE;
4
5 ALTER TABLE contains_module ADD CONSTRAINT FK_mid_contains_module
6 FOREIGN KEY (mid) REFERENCES module (mid)
7 ON DELETE CASCADE ON UPDATE CASCADE;
```

Auflistung 4.2: Fremdschlüsseldefinition in der Tabelle containsmodule

Die Tabelle *containsmodule* enthält demnach zwei Fremdschlüssel *FK_mgid_containsmodule* - welcher die Spalte *mgid* der Tabelle *modulegroup* referenziert - und *FK_mid_contains_module* - welcher die Spalte *mid* der Tabelle *module* referenziert. Wird der Wert eines Primärschlüssels in den Tabellen *modulegroup* oder *module* geändert oder gelöscht, werden die entsprechenden Zeilen in der Tabelle *contains_module* entsprechend geändert bzw. gelöscht.

4.1.2 Benutzerkontoverwaltung und Rechtevergabe

Benutzerkontoverwaltung und Rechtevergabe in MySQL

Mithilfe der *CREATE USER*-Anweisung können neue MySQL-Konten angelegt werden, jedoch nur wenn man die globale Berechtigung *CREATE USER* oder die Berechtigung *INSERT* für die Datenbank *mysql* besitzt, in deren Tabelle *user* alle Benutzerkonten gespeichert und verwaltet werden. Der Benutzername muss dabei immer nach dem Schema *<'Benutzerkontoname'@'Host'>* aufgebaut sein, wobei *(%)* bei der Hostangabe als universeller Platzhalter verwendet werden kann. Der in Auflistung 4.3 verwendete Befehl erzeugt beispielsweise ein neues Benutzerkonto mit dem Namen *anonymous* mit einem gleichnamigen zugehörigen Zugangspasswort.

```
1 CREATE USER 'anonymous'@'%' IDENTIFIED BY 'anonymous';
```

Auflistung 4.3: Erzeugung eines Benutzerkontos

Jedes neu angelegte Benutzerkonto erzeugt einen Eintrag in der Tabelle *user* der Datenbank *mysql*. Dieses Konto verfügt zu Beginn über keinerlei Rechte innerhalb des DBMS.

Mit der Anweisung *DROP USER* können bestehende Benutzerkonten gelöscht werden, wie z.B. in Auflistung 4.4 dargestellt.

```
1 DROP USER 'anonymous'@'%';
```

Auflistung 4.4: Löschen eines Benutzerkontos

MySQL bietet darauf aufbauend die Funktionalität existierenden Benutzerkonto bestimmte Rechte auf unterschiedlichen Ebenen zu gewähren bzw. zu entziehen. Dies wird mit der *GRANT*- bzw. der *REVOKE*-Anweisung durchgeführt.

Eine Übersicht aller in der Version 5.0.26 von MySQL definierten Rechte liefert das MySQL-Referenzhandbuch in Kapitel 13.5.1.3 *GRANT Syntax*[4]. Für das zu erstellende System sind dabei nur folgende Rechte relevant: [MyS07]:

SELECT — Erlaubt die Verwendung von SELECT-Anweisungen, d.h. das Abrufen von Informationen aus entsprechenden Datenbanktabellen.

INSERT — Erlaubt die Verwendung von INSERT-Anweisungen, d.h. das Einfügen von Tupeln in entsprechende Datenbanktabellen.

UPDATE — Erlaubt die Verwendung von UPDATE-Anweisungen, d.h. das Ändern von bereits bestehenden Tupeln in zugehörigen Datenbanktabellen.

DELETE — Erlaubt die Verwendung von DELETE-Anweisungen, d.h. das Löschen von Tupeln in den zugehörigen Datenbanktabellen.

EXECUTE — Erlaubt die Ausführung von sog. *gespeicherten Prozeduren*. Näheres dazu findet sich in Abschnitt 4.1.3.

Diese Rechte können den Benutzerkonten auf unterschiedlichen Ebenen gewährt bzw. entzogen werden[5]. Durch die Gegebenheiten an der Universität Koblenz-Landau sind nur folgende Ebenen relevant [MyS07]:

Datenbankebene — Datenbankberechtigungen gelten für alle Objekte in einer gegebenen Datenbank. Mit *GRANT <Recht> ON <Datenbankname>.** bzw. *REVOKE <Recht> ON <Datenbankname>.** können Datenbankberechtigungen gewährt bzw. entzogen werden.

Tabellenebene — Tabellenberechtigungen gelten für alle Spalten in einer gegebenen Tabelle. Mit *GRANT <Recht> ON <Tabellenname>.<Tabellenname>*

[4] `http://dev.mysql.com/doc/refman/5.0/en/grant.html` (abgerufen am 10.07.2007)
[5] `http://dev.mysql.com/doc/refman/5.0/en/grant.html` (abgerufen am 10.07.2007)

bzw. *REVOKE <Recht> ON <Datenbankname>.<Tabellenname>* können Datenbankberechtigungen gewährt bzw. entzogen werden.

Spaltenebene — Spaltenberechtigungen gelten für alle angegebenen Spalten in einer Datenbanktabelle. Sie können jedoch nur für die Rechte *SELECT*, *INSERT* und *UPDATE* vergeben werden.

Dabei muss beachtet werden, dass über das Webinterface lediglich die Datenbankebene abgedeckt wird. Möchte man Rechte auf Tabellen- und/oder Spaltenebene vergeben, müssen diese manuell nachgetragen werden.

Benutzerkontoverwaltung und Rechtevergabe für das zu erstellende System

Für das zu implementierende System werden die in Auflistung 4.5 aufgeführten Benutzerkonten für den Anonymen User, den Modulverantwortlichen, den Studiendekan und den Stundenplanersteller angelegt. Dabei ist zu beachten, dass die Bezeichner der Benutzerkonten maximal 16 Zeichen lang sein dürfen.

```
1 CREATE USER 'anonymous'@'%'
2 IDENTIFIED BY 'anonymous',
3 'moduleguarantor'@'%'
4 IDENTIFIED BY 'moduleguarantor',
5 'dean'@'%'
6 IDENTIFIED BY 'dean',
7 'ttablecreator'@'%'
8 IDENTIFIED BY 'timetablecreator';
```

Auflistung 4.5: Definitionen zur Benutzerkontoverwaltung

Hierbei fehlt ein Benutzerkonto für den Administrator der Datenbank. Der Administrator ist in diesem Fall zum Einen der Verantwortliche für die DBMS im Rechenzentrum der Universität Koblenz-Landau (zur Zeit Herr Krienke) und zum Anderen der Datenbankersteller.

Nachdem die Benutzerkonten über das Webinterface angelegt wurden, können diesen mit den in Auflistung 4.6 dargestellten *GRANT*-Befehlen die notwendigen Rechte vergeben werden. Da man als Datenbankersteller über das *GRANT*-Recht (*WITH GRANT OPTION*) verfügt und somit berechtigt ist, anderen Benutzerkonten seine eigenen Rechte zu übertragen, können diese Befehle z.B. über phpMyAdmin ausgeführt werden.

```
1 GRANT SELECT
2 ON websis.*
3 TO 'anonymous'@'%';
```

```
4
5  GRANT SELECT
6  ON websis.*
7  TO 'moduleguarantor'@'%';
8
9  GRANT INSERT, UPDATE, DELETE
10 ON websis.course
11 TO 'moduleguarantor'@'%';
12
13 GRANT INSERT, UPDATE, DELETE
14 ON websis.module
15 TO 'moduleguarantor'@'%';
16
17 GRANT INSERT, UPDATE, DELETE
18 ON websis.has_as_premise
19 TO 'moduleguarantor'@'%';
20
21 GRANT INSERT, UPDATE, DELETE
22 ON websis.has_literature
23 TO 'moduleguarantor'@'%';
24
25 GRANT INSERT, UPDATE, DELETE
26 ON websis.curriculum
27 TO 'moduleguarantor'@'%';
28
29 GRANT INSERT, UPDATE, DELETE
30 ON websis.coursetype_hoursperweek
31 TO 'moduleguarantor'@'%';
32
33 GRANT INSERT, UPDATE, DELETE
34 ON websis.is_competent_to_teach
35 TO 'moduleguarantor'@'%';
36
37 GRANT INSERT, UPDATE, DELETE
38 ON websis.consists_of_course
39 TO 'moduleguarantor'@'%';
40
41 GRANT SELECT
42 ON websis.*
43 TO 'dean'@'%';
44
45 GRANT INSERT, DELETE, UPDATE
46 ON websis.*
47 TO 'dean'@'%';
48
49 GRANT SELECT
50 ON websis.*
51 TO 'ttablecreator'@'%';
52
```

```
53 GRANT INSERT, DELETE, UPDATE
54 ON websis.courseinstance
55 TO 'ttablecreator'@'%';
56
57 GRANT INSERT, DELETE, UPDATE
58 ON websis.teaches
59 TO 'ttablecreator'@'%';
```

Auflistung 4.6: Definitionen zur Rechtevergabe

Alle Benutzerkonten können SELECT-Anweisungen auf der gesamten Datenbank (websis) ausführen.

Der Modulverantwortliche kann Einfüge-, Änderungs- und Löschoperationen auf den Tabellen *course*, *module*, *has_as_premise*, *has_literature*, *curriculum*, *coursetype_hoursperweek*, *is_competent_to_teach* und *consists_of_course* durchführen.

Der Studiendekan kann auf der gesamten Datenbank Einfüge-, Änderungs- und Löschoperationen durchführen.

Der Stundenplanersteller kann Einfüge-, Änderungs- und Löschoperationen auf der Tabelle *courseinstance* durchführen.

4.1.3 Gespeicherte Prozeduren und Funktionen

MySQL bietet die Möglichkeit sog. *gespeicherte Routinen* (*Prozeduren* und *Funktionen*) auf der Datenbank zu implementieren und zu speichern. Diese können aus *SQL-Anweisungen* und/oder *zusammengesetzten Anweisungen* bestehen, welche ihrerseits *Variablendeklarationen*, *Schleifen* und *Kontrollstrukturen* enthalten können, auf die in den nachfolgenden Abschnitten näher eingegangen wird [MyS07].

Der Client führt demnach keine SQL-Anweisungen durch, sondern ruft lediglich die auf der Datenbank gespeicherten Routinen auf. Kennt der Client deren genaue Signatur - sprich Bezeichner, Eingabetypen und evtl. Rückgabetypen - muss der Client über keinerlei Kenntnisse des Datenbankschemas verfügen, was jedoch bei der direkten Ausführung von SQL-Anweisungen unbedingt notwendig ist.

Gespeicherte Routinen werden v.a. in sicherheitskritischen Systemumgebungen (z.B. in Banken) verwendet. Dabei haben Benutzer keinen direkten Zugriff auf die Datenbank bzw. die Datenbanktabellen, da die Kommunikation lediglich über gespeicherte Routinen abläuft.

Einerseits kommt es durch den Einsatz von gespeicherten Routinen zu einer Performancesteigerung, da weniger Informationen zwischen Server und Client ausgetauscht werden müssen. Andererseits steigt jedoch die Belastung des DBMS, da hier der Großteil der Arbeitsschritte durchgeführt werden muss; der Client wird dabei entlastet. Im Falle des zu entwickelnden System dürfte sich die Belastung für das DBMS jedoch in Grenzen halten, da besonders arbeitsintensive (Einfüge-)Operationen eher selten durchgeführt werden.

Die Syntax zur Erstellung von gespeicherten Routinen entspricht dem *SQL:2003*-Standard [6], jedoch ist dieser auch in der aktuellsten Version von MySQL noch nicht vollständig implementiert worden.

Um gespeicherte Routinen erstellen zu können, benötigt man das *CREATE ROUTINE*-Recht. Das *ALTER ROUTINE*-Recht erlaubt es, Änderungen an bestehenden Routinen durchführen zu können [MyS07].

Gespeicherte Prozeduren

Eine gespeicherte Prozedur wird mit der *CREATE PROCEDURE*-Anweisung erstellt. Für eine Prozedur kann eine (auch leere) Parameterliste angegeben werden. Dabei können Parameter als *IN*, *OUT* oder *INOUT* charakterisiert werden, indem das entsprechende Schlüsselwort vor den Parameternamen gesetzt wird. *IN* ist die Standardeinstellung und bezeichnet Eingabewerte. Da MySQL für gespeicherte Prozeduren keine expliziten Rückgabewerte vorsieht, können Parameter mit *OUT* oder *INOUT* gekennzeichnet werden, so dass deren Werte nach der Ausführung der Prozedur vom Client abgerufen werden können. Für jeden Parameter muss der zugehörige Typ definiert werden, wobei alle in MySQL zulässigen Datentypen außer *COLLATE* erlaubt sind, die nur einen einzelnen Wert enthalten (Skalarwert) [MyS07].

Eine gespeicherte Prozedur kann in ihrem Prozedurkörper aus einer Menge von SQL-Anweisungen bestehen. Die Verwendung von zusammengesetzen Anweisungen ist ebenfalls erlaubt. Ein einfaches Beispiel für eine gespeicherte Prozedur findet sich in Auflistung 4.7.

```
1 CREATE PROCEDURE 'get_all_persons'(OUT _msgcode TINYINT,
2        OUT _msg TEXT, IN _pis_deleted BOOL)
3 BEGIN
4   SELECT *
5   FROM person
6   WHERE pis_deleted = _pis_deleted;
```

[6] http://www.sigmod.org/sigmod/record/issues/0403/E.JimAndrew-standard.pdf (abgerufen am 29.07.2007)

```
7
8     SET _msgcode = 1;
9     SET _msg = "The operation has been successfully executed.";
10 END;
```

Auflistung 4.7: Beispiel einer gespeicherten Prozedur

Die Prozedur *get_all_persons* enthält in ihrer Parameterliste zwei Ausgabeparameter *_msgcode* und *_msg* vom Typ *TINYINT* bzw. *TEXT* und einen Eingabeparameter *_pis_deleted* vom Typ *BOOL*. Die Prozedur liefert zum Einen durch die verwendete *SELECT*-Anweisung eine Ergebnismenge zurück, welche - abhängig vom Wert des Parameters *_pis_deleted* - lediglich alle als gelöscht oder alle als nicht gelöscht markierten Personen enthält. Zum Anderen kann nach Ausführung der Prozedur auf die Werte der beiden Ausgabeparameter - wie im nächsten Abschnitt beschrieben - zugegriffen werden.

Enthält demnach eine gespeicherte Prozedur eine SQL-Anweisung, welche eine Ergebnismenge zurückliefert, stellt diese die Rückgabe der Prozedur dar. Dabei wird lediglich die erste ermittelte Ergebnismenge zurückgeliefert; alle folgenden Ergebnismengen werden ignoriert. Jedoch endet die Prozedurausführung nicht mit dem Ermitteln der ersten Ergebnismenge, so dass auch der nachfolgende Code ausgeführt wird.

Das Aufrufen von gespeicherten Prozeduren

Damit ein Benutzerkonto eine auf der Datenbank gespeicherte Prozedur aufrufen kann, muss diesem das *EXECUTE*-Recht gewährt werden. Dem Ersteller einer gespeicherten Prozedur wird dieses Recht und zusätzlich das *ALTER ROUTINE*-Recht automatisch zugewiesen [MyS07].

Aufgerufen werden gespeicherte Prozeduren mit Hilfe der *CALL*-Anweisung. Ein entsprechender beispielhafter Aufruf für die in Auflistung 4.7 definierte Prozedur, welche alle als gelöschte markierte Datensätze in der Personentabelle ermittelt, findet sich in Auflistung 4.8.

```
1 CALL get_all_persons(@_msgcode, @_msg, true);
```

Auflistung 4.8: Beispiel für den Aufruf einer gespeicherten Prozedur

Für als *OUT* charakterisierte Parameter muss beim Aufruf ein @-Zeichen als Präfix verwendet werden. Um nach Ausführung der Prozedur auf den Wert eines Ausgabeparameters zugreifen zu können, muss der Client eine entsprechende

SELECT-Anweisung durchführen. Um im obigen Beispiel an den Wert des Ausgabeparameters *_msg* zu gelangen, würde diese wie in Auflistung 4.9 lauten.

```
1 SELECT @_msg;
```

Auflistung 4.9: Beispiel für die Werteermittlung von Ausgabeparametern

Gespeicherte Funktionen

Mit der *CREATE FUNCTION*-Anweisung werden gespeicherte Funktionen erstellt. Im Gegensatz zu gespeicherten Prozeduren besitzen gespeicherte Funktionen einen zu definierenden Rückgabewert. Sie können jedoch nicht per *CALL*-Anweisung durch den Client aufgerufen werden, sondern können nur innerhalb von gespeicherten Prozeduren oder in entsprechenden SQL-Anweisungen verwendet und integriert werden. Eine (auch leere) Parameterliste kann hier ebenso angegeben werden, allerdings existieren dabei nur Eingabeparameter, d.h. dass alle Parameter implizit mit *IN* charakterisiert sind [MyS07]. In Auflistung 4.10 findet sich ein Beispiel für eine gespeicherte Funktion.

```
1 CREATE FUNCTION `get_extended_token`(_token VARCHAR(8))
2        RETURNS VARCHAR(10) CHARSET latin1
3 BEGIN
4   RETURN CONCAT(_token, "TP");
5 END;
```

Auflistung 4.10: Beispiel einer gespeicherten Funktion

Die gespeicherte Funktion *get_extended_token* ergänzt ein Kürzel durch das Suffix „TP“ und gibt das erweiterte Kürzel zurück. Sie besitzt einen Eingabeparameter *_token* vom Typ *VARCHAR(8)*. Der Rückgabewert ist vom Typ *VARCHAR(10)*. Rückgabewerte, dessen Datentyp ebenfalls nur Skalarwerte enthalten dürfen, werden in gespeicherten Funktionen durch das Schlüsselwort *RETURN* definiert.

In gespeicherten Funktionen dürfen keine SQL-Anweisungen verwendet werden, die eine Ergebnismenge zurückgeben, wie z.B. die SELECT-Anweisung.

Zusammengesetzte Anweisungen

BEGIN...END-Umgebungen in gespeicherten Routinen dienen zur Zusammenfassung von Anweisungslisten mit eigenem lokalen Namensraum. Die einzelnen Anweisungen werden dabei durch ein Semikolon *(;)* getrennt [MyS07].

In solchen Umgebungen können die im Folgenden erläuterten und bei der Implementierung der gespeicherten Routinen letztlich genutzten Anweisungen verwendet werden.

Variablendeklaration und Wertezuweisung

Mit der *DECLARE*-Anweisung können lokale Variablen deklariert werden, deren Gültigkeit sich auf die *BEGIN...END*-Umgebung beschränkt, in der sie definiert worden sind. Mit dem Schlüsselwort *DEFAULT* kann einer lokalen Variablen ein Anfangswert zugeordnet werden. Zusätzlich muss für jede Variable ein Typ angegeben werden, der wiederum nur Skalarwerte speichern darf. Ein Beispiel für eine lokale Variablendeklaration ist in Auflistung 4.11 gezeigt [MyS07].

```
1 DECLARE _creditpoints TINYINT DEFAULT 10;
```

Auflistung 4.11: Beispiel einer Variablendeklaration

Mit der *SET*-Anweisung kann einer bereits deklarierten lokalen Variablen oder einem Eingabe- bzw. Ausgabeparameter ein Wert zugewiesen werden. Auflistung 4.12 zeigt ein zugehöriges Beispiel.

```
1 SET _creditpoints = 100;
```

Auflistung 4.12: Beispiel für eine Wertezuweisung an eine Variable

Die SELECT...INTO-Anweisung

Mit der *SELECT...INTO*-Anweisung werden die Werte der in der *SELECT*-Anweisung ausgewählten Spalten direkt in die nach dem Schlüsselwort *INTO* angegebenen bereits deklarierten Variablen gespeichert. Es kann dabei aber lediglich die erste Zeile der Ergebnismenge abgerufen und gespeichert werden [MyS07]. Ein Beispiel für die Verwendung der *SELECT...INTO*-Anweisung findet sich in Auflistung 4.13.

```
1 CREATE FUNCTION `get_creditpoints_for_modulegroup`
2        (_mgid VARCHAR(8)) RETURNS TINYINT
3 BEGIN
4   DECLARE _mgcreditpoints TINYINT DEFAULT 0;
5
6   SELECT mgcreditpoints INTO _mgcreditpoints
7   FROM modulegroup
8   WHERE mgid = _mgid;
```

```
9
10    RETURN _mgcreditpoints;
11  END;
```

Auflistung 4.13: Beispiel für die Verwendung der SELECT...INTO-Anweisung

Hierbei wird die Leistungspunktezahl *(mgcreditpoints)* für die Modulgruppe mit der übergebenen Id *(mgid)* in die Variable *_mgcreditpoints* gespeichert und anschließend zurückgegeben.

Die Verwendung der CURSOR-Anweisung

MySQL erlaubt in gespeicherten Prozeduren nicht nur die Rückgabe von Ergebnismengen mit der *SELECT*-Anweisung, sondern auch deren lokale Speicherung, um diese prozedurintern verarbeiten zu können. Für die Speicherung und die iterative Bearbeitung des Ergebnisses werden dabei sog. *Cursor* verwendet [MyS07].

Auflistung 4.14 zeigt die Deklaration eines Cursors mit dem Bezeichner *_cur1*. Dieser speichert als Ergebnismenge die Lehrkapazität aller Personen, die in der Tabelle *person* gespeichert sind. Nach dem Schlüsselwort *FOR* muss dabei immer eine *SELECT*-Anweisung folgen (in diesem Fall *SELECT pteachingcapacity FROM person*), welche eine zu speichernde Ergebnismenge liefert.

```
1  DECLARE _cur1 CURSOR FOR SELECT pteachingcapacity FROM person;
```

Auflistung 4.14: Beispiel für die Deklaration eines Cursors

Cursor können auch in gespeicherten Funktionen verwendet werden, da die durch die *SELECT*-Anweisungen gelieferten Ergebnismengen nicht direkt zurückgegeben, sondern lokal in einer Variablen gespeichert werden.

Mit der *OPEN*-Anweisung wird ein zuvor deklarierter Cursor geöffnet und initialisiert. Für den bereits deklarierten Cursor, würde der Befehl wie in Auflistung 4.15 lauten.

```
1  OPEN _cur1;
```

Auflistung 4.15: Beispiel für die Initialisierung eines Cursors

Die *CLOSE*-Anweisung schließt einen zuvor initialisierten Cursor, wie in Auflistung 4.16 dargestellt.

```
1 CLOSE _cur1;
```

Auflistung 4.16: Beispiel für das Schliessen eines Cursors

Die WHILE-Anweisung zum Durchlaufen eines Cursors

Um auf die einzelnen Datensätze der Ergebnismenge zugreifen zu können, muss der definierte Cursor auf den jeweils folgenden Datensatz verschoben werden. Dies lässt sich z.B. mit der *WHILE*-Anweisung durchführen [MyS07]. Diese wird mit einer *WHILE <Bedingung> DO <Anweisungsfolge> END WHILE*-Umgebung definiert.

Die Beeinflussung der Schleifenbedingung

Die *<Bedingung>* kann dabei z.B. mit dem sog. *CONTINUE*-Handler beeinflusst und demnach verändert werden. Dieser erlaubt es Anweisungen durchführen zu lassen, wenn eine bestimmte Bedingung erfüllt oder ein bestimmter Zustand erreicht wird. *CONTINUE* bedeutet hier, dass die Ausführung der aktuellen Routine nach der Ausführung der Handler-Anweisung weiterläuft.[MyS07]

Auflistung 4.17 zeigt die Deklaration eines solchen *CONTINUE*-Handlers. Sobald ein bestimmter Zustand während des Programmablaufs (hier *SQLSTATE '02000'*) erreicht wurde, wird die nachfolgende Anweisung ausgeführt (hier *SET _has_more_rows = FALSE*).

```
1 DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
2 SET _has_more_rows = FALSE;
```

Auflistung 4.17: Beispiel für die Deklaration eines CONTINUE-Handlers

Das Zusammenspiel von CURSOR-, HANDLER- und WHILE-Anweisung

Auflistung 4.18 zeigt ein Beispiel, welches die CURSOR-, HANDLER- und WHILE-Anweisung verwendet, um durch eine Ergebnismenge zu iterieren und somit auf deren einzelnen Elemente zugreifen zu können.

```
1 CREATE FUNCTION `get_aggregated_teachingcapactity`() RETURNS INT
2 BEGIN
3   DECLARE _has_more_rows BOOL DEFAULT TRUE;
```

```
4   DECLARE _teachingcapacity TINYINT DEFAULT 0;
5   DECLARE _aggr_teachingcapacity INT DEFAULT 0;
6   DECLARE _cur1 CURSOR FOR SELECT pteachingcapacity
7
8   DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
9   SET _has_more_rows = FALSE;
10
11  OPEN _cur1;
12
13  FETCH _cur1 INTO _teachingcapacity;
14  WHILE _has_more_rows DO
15    SET _aggr_teachingcapacity = _aggr_teachingcapacity
16                                 + _teachingcapacity;
17
18    FETCH _cur1 INTO _teachingcapacity;
19  END WHILE;
20
21  CLOSE _cur1;
22
23  RETURN _aggr_teachingcapacity;
24 END;
```

Auflistung 4.18: Beispiel für die Durchlaufen eines Cursors mit der WHILE-Anweisung

Vor dem Betreten der *WHILE*-Schleife wird mit der Anweisung *FETCH _cur1 INTO _teachingcapacity;* die erste Zeile der Ergebnismenge ermittelt. Somit wird sichergestellt, dass die *WHILE*-Anweisung gar nicht erst betreten wird, falls die Ergebnismenge leer ist. Die *FETCH <Cursorname> INTO <Variable(n)>*Anweisung, welche auch innerhalb der *WHILE*-Schleife verwendet wird, ermittelt - analog zur *SELECT <Spalte(n)>INTO<Variable(n)>*-Anweisung - die nächste Zeile der Ergebnismenge. Die einzelnen Spalten einer Zeile werden in die entsprechenden nach dem Schlüsselwort *INTO* angegebenen Variablen gespeichert.

Sind alle Zeilen der Ergebnismenge durchlaufen worden, wird ein Zustand erreicht, welcher dem *SQLSTATE '02000'* entspricht, wodurch der Wert der Variablen *_has_more_rows* durch den definierten *CONTINUE*-Handler auf den Wert *FALSE* gesetzt wird. Dies führt zum Verlassen der *WHILE*-Schleife, da deren Bedingung nicht mehr erfüllt ist. Dies bedeutet für das Beispiel, dass solange mit dem Durchlaufen des *Cursors* fortgefahren wird, bis ein Zustand erreicht wird, der dem *SQLSTATE '02000'* entspricht.

Die Funktion *get_aggregated_teachingcapactity* berechnet demnach die Summe aller gespeicherten Lehrkapazitäten in der Tabelle *person* und gibt diese zurück.

Die IF-Anweisung

Mit der IF-Anweisung können abhängig von einer Bedingung unterschiedliche Anweisungsfolgen ausgeführt werden. Ein Beispiel hierfür findet sich in Auflistung 4.19.

```
1  CREATE PROCEDURE `get_all_persons`(IN _all BOOL, IN _pis_deleted
2  BOOL)
3  BEGIN
4    IF _all = FALSE THEN
5      SELECT pid, pfirstname, psurname, pinstitute
6      FROM person
7      WHERE pis_deleted = _pis_deleted;
8    ELSE
9      SELECT pid, pfirstname, psurname, pinstitute
10     FROM person;
11   END IF;
12 END
```

Auflistung 4.19: Beispiel für die Verwendung der IF-Anweisung

Wenn beim Aufruf der Prozedur *get_all_person* der Wert des Eingabeparameters *_all FALSE* ist, werden - abhängig vom Wert des zweiten Eingabeparameters *_pis_deleted* - lediglich alle gelöschten oder nicht gelöschten Personen zurückgegeben. Ist der Wert von *_all* jedoch *true*, werden alle Personen zurückgegeben, welche in der Tabelle *person* gespeichert sind.

Implementierung der gespeicherten Routinen auf der Datenbank

Wie bereits erwähnt, benötigt man für die Erstellung von gespeicherten Routinen das *CREATE ROUTINE*-, zum Verändern von bestehenden Routinen das *ALTER ROUTINE*-Recht. Da diese nicht bei der Rechtevergabe über das Webinterface der Universität Koblenz-Landau vergeben werden können, muss das Rechenzentrum diese nachträglich für die jeweiligen Benutzerkonten auf der entsprechenden Datenbank setzen.

Dem Ersteller der gespeicherten Routinen wird das für deren Aufruf benötigte *EXECUTE*-Recht automatisch zugewiesen. Dieser kann - unter der Voraussetzung, dass er selbst über das Recht verfügt, anderen Benutzerkonten seine eigenen Rechte zu übertragen (*WITH GRANT OPTION*) - weiteren auf der Datenbank definierten Benutzerkonten das *EXECUTE*-Recht mittels der *GRANT*-Anweisung zuweisen. Für die zu implementierende Datenbank müssen also die in Auflistung 4.20 dargestellten *GRANT*-Anweisungen durchgeführt werden.

```
1  GRANT EXECUTE
2  ON websis.*
3  TO 'anonymous'@'%';
4
5  GRANT EXECUTE
6  ON websis.*
7  TO 'moduleguarantor'@'%';
8
9  GRANT EXECUTE
10 ON websis.*
11 TO 'dean'@'%';
12
13 GRANT EXECUTE
14 ON websis.*
15 TO 'ttablecreator'@'%';
```

Auflistung 4.20: Vergabe des EXECUTE-Rechts

Bei der Ausführung einer gespeicherten Prozedur stellt sich die Frage, ob das jeweilige aufrufende Benutzerkonto über ausreichende Rechte verfügt und verfügen soll, um die in der gespeicherten Prozedur enthaltenen SQL-Anweisungen ausführen zu können. Dieses Problem muss bereits bei der Implementierung der gespeicherten Prozeduren durch den Ersteller geregelt werden. Dieser hat die Möglichkeit nach dem Prozedurrumpf mit dem Schlüsselwort *SQL SECURITY* festzulegen, ob beim Aufruf einer gespeicherten Prozedur die Rechte des Prozedurerstellers (*DEFINER*) oder des aufrufenden Benutzerkontos (*INVOKER*) gelten sollen.

Im Rahmen dieser Studienarbeit muss bei jeder gespeicherten Routine nach dem Rumpf *SQL SECURITY INVOKER* angegeben werden, um sicherzustellen, dass Benutzerkonten nur solche SQL-Anweisungen ausführen können, für welche diese über entsprechende Rechte verfügen.

Mit der somit im System definierten Rechtevergabe ist es demnach allen Benutzerkonten erlaubt, gespeicherte Prozeduren aufzurufen. Falls diese SQL-Anweisungen enthalten, für welche das aufrufende Benutzerkonto nicht über das entsprechende Recht verfügt, wird eine entsprechende Fehlermeldung von der Datenbank zurückgegeben (<*Anweisungsname*> *command denied to user* <*Benutzerkontoname*> *for table* <*Tabellenname*>). Benutzerkonten ist es zudem erlaubt, SQL-Anweisungen direkt durchzuführen, für welche sie das zugehörige Recht besitzen.

Für die Implementierung der gespeicherten Routinen wurde der *MySQL Query Browser* in der Version 1.2.12 verwendet, welcher Teil der *MySQL GUI Tools* in

der Version 5.0[7] ist.

Konventionen für Parameterlisten bei gespeicherten Prozeduren

Verfügt eine gespeicherte Prozedur über eine nicht-leere Parameterliste, werden bestimmte Konventionen in der Reihenfolge der anzugebenden Parameter eingehalten, um den Prozeduraufruf und die Ergebnisverarbeitung in der Anwendungslogik vereinheitlichen zu können.

Zunächst werden die Ausgabeparameter - sprich die mit *OUT* charakterisierten Parameter - angegeben, gefolgt von solchen, die als *INOUT* definiert sind. Zuletzt werden die Eingabeparameter bzw. die als *IN* charakterisierten Parameter notiert.

Konventionen bei der Bezeichnervergabe in gespeicherten Routinen

Bei der Bezeichnervergabe von gespeicherten Routinen wird der Unterstrich (_) als Trennzeichen verwendet. Alle Parameter und alle in gespeicherten Routinen deklarierte Variablen beginnen mit einem Unterstrich. Dies ist v.a. hilfreich um die Eindeutigkeit von Bezeichnern zu erhalten, wenn ein Parameter oder eine Variable den gleichen Namen wie eine Tabellenspalte in der Datenbank besitzt.

Meldungen nach dem Ausführen von gespeicherten Prozeduren

Alle Prozeduren, die keine Ergebnismenge zurückliefern, enthalten in ihrer Parameterliste zwei Ausgabeparameter *_msgcode* vom Typ *TINYINT* und *_msg* vom Typ *TEXT*. *_msgcode* gibt einen Hinweis darauf, ob eine bestimmte Aktion erwartungsgemäß ausgeführt wurde oder warum eine Aktion nicht erfolgreich war. Dabei lassen sich vier Meldungscodes unterscheiden:

- *0* bedeutet, dass ein Datensatz, der eingefügt werden soll, bereits existiert oder keine Änderungen an einem bestimmten Datensatz vorgenommen worden sind.

- Mit der *1* wird signalisiert, dass die gewünschte Aktion erfolgreich durchgeführt wurde.

[7] `http://dev.mysql.com/downloads/gui-tools/5.0.html` (abgerufen am 29.07.2007)

- *2* bedeutet, dass eine übergebene Id nicht gefunden werden konnte.

- *3* steht für einen unbekannten Fehler. Um eine detaillierte Fehlerangabe zu erhalten, müssen entsprechende Funktionen in der MySQL-API der Anwendungslogik verwendet werden.

Der Ausgabeparameter *_msg* enthält die zum *_msgcode* zugehörige textuelle Meldung.

AUTOCOMMIT in gespeicherten Prozeduren

Empfehlenswert und notwendig ist es den *AUTOCOMMIT*-Modus der Datenbank in der Anwendungslogik auszuschalten. Ist dieser angeschaltet, werden alle durch SQL-Befehle ausgelösten Änderungen nach der Befehlsausführung automatisch und irreversibel übernommen, so dass das *ACID*-Paradigma unzureichend berücksichtigt wird, da ausgeführte Änderungen nicht rückgängig gemacht werden können.

Aufgrund von Problemen in einigen Versionen von MySQL, welche dazu führen, dass innerhalb von gespeicherten Prozeduren die durch den Client gesetzte *AUTOCOMMIT*-Einstellung für die Datenbankverbindung ignoriert wird, wurde sicherheitshalber in jeder implementierten gespeicherten Prozedur manuell der Befehl *SET AUTOCOMMIT = 0;* eingefügt, um sicherzustellen, dass Änderungen erst dann in die Datenbank übernommen werden, wenn explizit ein *COMMIT*-Befehl vom Client durchgeführt wird.

Probleme mit Gespeicherten Prozeduren und PHP

Damit gespeicherte Prozeduren mit PHP aufgerufen werden können, muss die *MySQL Improved Extension*[8] verwendet werden, die erst ab PHP 5.0 verfügbar ist. Hierbei muss jedoch beachtet werden, dass zwischen zwei Prozeduraufrufen der durch eine Prozedur belegte Speicher mittels der *mysqli_free_result*-Funktion freigegeben werden muss, wenn eine Ergebnismenge zurückgeliefert wird.

Die Standard MySQL-Bibliothek für PHP[9] kann nicht für das Aufrufen und die Verarbeitung von gespeicherten Prozeduren verwendet werden, da der Fehler „Commands out of sync; you can't run this command now“ ausgelöst wird, wenn

[8] `http://de3.php.net/manual/en/ref.mysqli.php` (abgerufen am 29.07.2007)
[9] `http://de3.php.net/manual/en/ref.mysql.php` (abgerufen am 29.07.2007)

während einer Datenbankverbindung mehr als eine gespeicherte Prozedur aufgerufen wird.

4.1.4 Das Befüllen der Datenbanktabellen

Nachdem das Datenbankschema implementiert und die gespeicherten Routinen auf der Datenbank implementiert sind, müssen die für das Modulhandbuch relevanten Daten in die Datenbank eingetragen werden.

Diese mussten größtenteils aus der Prüfungsordnung[10] und dem (Online-)Modulhandbuch [11] extrahiert werden.

Die relevanten Daten des Online-Modulhandbuchs, welches in einem Wiki umgesetzt ist, wurde mit Hilfe eines PHP-Skripts geparst und in eine Textdatei extrahiert. Diese enthält die notwendigen Prozeduraufrufe, um die extrahierten Daten in der Datenbank speichern zu können. Informationen, die nicht automatisiert geparst bzw. extrahiert werden konnten, wurden manuell in das PHP-Skript integriert.

Letztlich wurde eine SQL-Datei erstellt, welche die SQL-Anweisungen zum Implementieren des Datenbankschemas, die Definition der gespeicherten Routinen und die Prozeduraufrufe zur Speicherung der relevanten Daten enthält. Diese SQL-Datei wurde mit dem *MySQL-Browser* auf der entsprechenden Datenbank ausgeführt.

4.2 Webseitenimplementierung

4.2.1 Zope

Einführung in Zope

Zope[12] ist ein Webanwendungsserver[LPMS]. Dieser erlaubt Entwicklern mit unterschiedlicher Erfahrung das Erstellen von Webanwendungen. Statt statischer

[10] `http://www.uni-koblenz.de/~akpramt/index.php?show=ordnungen` (abgerufen am 29.07.2007)
[11] `http://helena.uni-koblenz.de/~ist/istwiki/index.php/Modulhandbuch` (abgerufen am 29.07.2007)
[12] `http://www.zope.org` (abgerufen am 15.10.2007)

HTML Seiten werden dynamische Seiten generiert, welche unabhängig vom Design sind.

Ein weiterer Vorteil von Zope ist es, dass sich auch das Design der Webseiten schnell ändern lässt, da es lediglich zentral an einem Ort verändert werden muss. Bei statischen HTML Seiten muss hingegen jede einzelne Seite manuell aktualisiert werden. Dies ist wiederum sehr fehleranfällig und zeitaufwändig, bedenkt man, dass bei größeren Internetprojekten die Anzahl der Seiten in die Tausende gehen kann.

Um den Inhalt einer Webseite zu erzeugen wird eine einfache Programmiersprache (C, Perl, Python etc.) verwendet. Es empfiehlt sich hier Python zu verwenden, da Zope selbst in dieser Sprache entwickelt wurde. Darüber hinaus bietet Zope die Funktionalität zum Erstellen von Vorlagen, ein Sicherheitsmodell, Persistenz von Daten, Unterstützung von Sessions und vieles mehr.

Zope ermöglicht es auch Addons zu entwickeln, wie etwa ein Content Management System (CMS), wie es an der Universität Koblenz (Zope-Version: 2.6.1; Python-Version: 2.1.3 [Stand: 15.10.2007]) verwendet wird.

Zope Konzepte

Zope ist ein Framework, welches für den Entwickler viele Detailfragen übernimmt, wie etwa zur Persistenz und Integrität von Daten oder der Zugriffskontrolle.

Das Framework stellt Werkzeuge zur Verfügung, welches es erlauben Webanwendungen schneller als mit anderen Sprachen oder Frameworks zu entwickeln [LPMS]. Des Weiteren erlaubt Zope das Schreiben von Anwendungslogik in Python und bietet einen Addon-Support für Perl.

Objektorientierung — Im Gegensatz zu ASP oder älteren Versionen von PHP (älter als Version 5) basiert Zope auf dem Konzept der Objektorientierung.

Bei Zope ist alles im Web weitestgehend objektorientiert, bei einigen Konzepten weicht Zope jedoch von der Idee der Objektorientierung ab. Statt Vererbung unterstützt Zope Akquisition, welche zusätzlich in den Containern eines Objekts nach den entsprechenden Methoden sucht. Ein Container ist ein übergeordnetes Objekt, welches weitere Objekte beinhaltet. So ist Beispielsweise der Container eines „DTML Method“ Zope-Objekts in einem „DTML_Example folder“ Zope-Objekt das „DTML_Example folder“ Zope-Objekt.

Eine URL ist bei Zope ein Pfad zu einem Objekt in einer Menge von Containern und das HTTP-Protokoll ist ein Weg, um Nachrichten an das Objekt zu senden und die Antworten zu empfangen. Die Objektstruktur bei Zope ist hierarchisch aufgebaut. Eine Zope Seite besteht aus Objekten, welche weitere Objekte enthalten können. Selbst die URLs passen in diesen Objektkontext hinein. So ist die URL `/Marketing/index.html` nichts weiter, als ein Datei-Objekt „`index.html`“ in einem Ordner-Objekt „`Marketing`“.

Objektpublizierung — Die bei der Verarbeitung erstellten Objekte werden publiziert. Dies geschieht sequenziell:

1. Der Webbrowser sendet eine Anfrage an den Zope-Server. Diese hat die Form `protocol://host:port/path?querystring`
   z.B. `http://www.zope.org:8080/Resources?batch_start=100`

2. Zope zerlegt die URL in ihre einzelnen Bestandteile

   - *host* := `http://www.zope.org`
   - *port* := 8080
   - *path* := `/Resources`
   - *querystring* := `?batch_start=100`

3. Zope lokalisiert das Objekt, welches dem *path* entspricht in der Objektdatenbank.

4. Zope führt das Objekt aus und benutzt dabei den *querystring* als eine Liste von Parametern, welche das Verhalten des Objektes beeinflussen können.

5. Nach dem Ausführen wird ein Wert an den Browser zurückgegeben. Üblicherweise handelt es sich dabei um HTML, Datei- oder Bilddaten.

6. Die Daten werden vom Browser interpretiert und angezeigt.

Through-The-Web Management — Um Objekte zu erzeugen und mit ihnen zu arbeiten, stellt Zope ein Webinterface zur Verfügung. Das komplette Management von Zope kann über dieses Interface geschehen. Man kann beispielsweise neue Zope-Objekte anlegen und sogar völlig neue Objekte definieren. Diese Objekte können überall in der Objekthierarchie abgelegt werden.

Der Benutzer kann über Tabs zwischen den verschiedenen Sichten eines Objekts wechseln. Diese unterscheiden sich je nach Art des Objekts. Beispielsweise hat ein „DTML Method“ Zope-Objekt ein Edit-Tab, welches es erlaubt die Dokumentsourcen zu editieren. Ein „Database Connection“ Zope-Objekt hingegen stellt Sichten zum Modifizieren der Verbindungseinstellungen zur Verfügung.

Des Weiteren haben alle Objekte einen Security Tab, welcher vielfältige Sicherheitseinstellungen ermöglicht. An der Universität Koblenz kann dieses Webinterface aufgerufen werden, indem man dem *path* den Suffix `/manage` hinzufügt.

Sicherheit — Zope wurde von Anfang an für die Arbeit mit mehreren Benutzern (Zope-Admin, Datenbank-Admin, Web-Admin etc.) erstellt. Daher bietet Zope sehr feingranulare Einstellungsmöglichkeiten, welche sich von Objekt zu Objekt unterscheiden, was die Zugriffskontrolle betrifft. Man kann z.B. dem Web-Admin erlauben „SQL Methods“ Objekte zu benutzen, ihm jedoch das Erstellen, Editieren und sogar das Anzeigen des Quellcodes von diesem verweigern. Ebenfalls lässt es sich einrichten, dass ein User nur bestimmte Objekte, wie zum Beispiel Ordner- oder DTML-Objekte verwenden kann.

Zope bietet die Möglichkeit, Benutzer durch so genannte „User Folder“ zu managen. Dies sind spezielle Ordner, welche die Information über einen Benutzer beinhalten. Es gibt einige Zope-Addons, welche erweiterte Typen der „User Folder“ anbieten. Mit diesen ist es möglich, die benötigten Daten aus externen Quellen, wie etwa einer Datenbank oder einem LDAP-Server, zu holen. Die Fähigkeit „User Folder“ anzulegen kann an Benutzer für bestimmten Unterordner weitergegeben werden. Dadurch hat man die Möglichkeit die Kontrolle über Bereiche der Webseite an andere Benutzer zu übertragen ohne das Risiko einzugehen, dass diese an den Überordnern etwas ändern können.

Natürliche Objektpersistenz und Transaktionen — Die Zope-Objekte werden in einer objektorientierten Datenbank gespeichert, der Zope Object Database (ZODB). Jede Webanfrage wird dabei als eine eigene Transaktion behandelt und bei einem Fehler werden die Änderungen sofort wieder rückgängig gemacht. Die Datenbank bietet ausserdem einen „Multi-Level-Undo“, welcher dem Zope-Admin erlaubt, Änderungen mit einem Klick rückgängig zu machen.

Die Details über die Persistenz und die Transaktionen sind für die Anwendungsentwickler transparent gehalten. Ebenfalls lassen sich relationale Datenbanken, welche mit Zope verwendet werden, in das Zope-Transaktions-Framework einbinden.

Akquisition — Zope-Objekte können nicht nur von ihren Superklassen die Attribute und das Verhalten erben, sondern auch von ihren Containern. Zope, geht dabei so vor, dass wenn ein Objekt nicht in der Vererbungshierarchie zu finden ist, die Aquisitionshierarchie durchsucht wird. Dies funktioniert mit allen Zope-Objekten und ist ein mächtiger Weg um Ressourcen zu zentralisieren.

Beispielsweise lässt sich so eine SQL Query in einem Objekt definieren, welche dann auch von allen Unterobjekten verwendet werden kann. Muss die Query mal geändert werden, so muss diese Änderung nur in dem obersten Objekt, welches diese Query benutzt geändert werden. Auf diese Weise kann man auch das Design der Website für alle Unterordner vereinheitlichen. Will man bei einer Webseite z.B. in der Unterkategorie „Sport“ ein zum Rest der Seite unterschiedliches Design haben, so definiert man im Ordner-Objekt das Design und vererbt es an alle weiteren Webseiten unterhalb dieses Ordners. So ist das Design für die gesamte Kategorie „Sport“ gleich und muss nur an einer Stelle modifiziert werden, falls das Design geändert wird.

Erweiterbarkeit — Zope bietet viele Werkzeuge, die das Schreiben von Erweiterungen erleichtern und es existieren bereits viele Erweiterungen (z.B. XML- Tools, e-Commerce Integration, Bug Tracker, Newsletter-Systeme). Viele dieser Erweiterungen sind Open Source und werden von der Zope-Community gepflegt.

Besonderheiten bei der Integration von PHP-Skripten in Zope

Generell ist Zope nicht für die Arbeit mit PHP geschaffen, aber es gibt hierfür ein Addon von Wei He, den PHParser[13].

Dieser basiert auf der „DTML-Document-Class“, was die Nutzung von FTP, WebDAV uvm. ermöglicht. Der PHParser arbeitet als Postprozessor, d.h. erst nachdem alle DTML-Tags des Dokuments verarbeitet wurden, wird das Ergebnis durch einen externen PHP-Interpreter geschickt. Dabei wird eine komplette PHP-Umgebung emuliert, welche es ermöglichen soll PHP-Skripte ohne Änderungen sowohl innerhalb, als auch außerhalb von Zope auszuführen [He07].

Die globalen Variablen von PHP werden dabei durch den PHParser an Zope weitergeleitet. Dies ist allerdings für die Variablen von PHP 4 der Fall. WebSIs jedoch ist in PHP5 geschrieben, wo die Variablennamen geändert wurden (`$HTTP_GET_VARS` ⇒ `$_GET`, `$HTTP_POST_VARS` ⇒ `$_POST`, `$HTTP_SERVER_VARS` ⇒ `$_SERVER`

[13] `http://www.zope.org/Members/hewei/PHParser`

etc.). Diese Inkompatibilität zu PHP 5 musste im Nachhinein beseitigt werden. Dies wurde dadurch erreicht, dass Zope statt PHP das Wrapper-Shell-Skript in Auflistung 4.21 aufruft. Dort werden die Variablen für PHP 5 gesetzt und danach der PHP-Interpreter gestartet.

```
ssh penguin2.uni-koblenz.de /usr/bin/env
    REQUEST_URI=\"$REQUEST_URI\"
    MY_DOCUMENT_ROOT=\"$DOCUMENT_ROOT\" ZOPE_USER=\"$ZOPE_USER\"
    HTTP_COOKIE=\"$HTTP_COOKIE\" QUERY_STRING=\"$QUERY_STRING\"
    REQUEST_METHOD=$REQUEST_METHOD
    /usr/bin/php5 -F "$PATH_TRANSLATED"
```

Auflistung 4.21: Wrapper-Shell-Skript zum hinzufügen der Variablen

Diese Variablen werden in der zentralen `index.php5`-Datei in ihre jeweiligen Variablen geschrieben. Dafür ist die Methode `setRequestVariable()` aus Auflistung 4.22 zuständig.

```
function setRequestVariable($content, &$requestVariable) {
  $query = $content;
  $a = explode('&', $query);
  $i = 0;
  while ($i < count($a)) {
    $b = split('=', $a[$i]);
    $requestFieldName = htmlspecialchars(urldecode($b[0]));
    if(strpos($requestFieldName, "[]") !== false) {
      if(!is_array($requestVariable[str_replace("[]",
                                          "",
                                          $requestFieldName)])){
        $requestVariable[str_replace("[]",
                                     "",
                                     $requestFieldName)] = array();
      }
      array_push($requestVariable[str_replace("[]",
                                      "",
                                      $requestFieldName)],
                 htmlspecialchars(urldecode($b[1])));
    } else {
      $requestVariable[$requestFieldName] = htmlspecialchars(
                                              urldecode($b[1]));
    }
    $i++;
  }
}
```

Auflistung 4.22: Funktion zum Speichern der Zope-Variablen in den globalen PHP5-Variablen

Diese Funktion schreibt die jeweilige Zope-Variable (`$content`) in die zugehörige globale PHP-Variable (`&$requestVariable`). Diese Funktion wird für die GET- bzw. POST-Variable nur aufgerufen, wenn die Zope-Variablen mindestens einen Wert enthalten (siehe Auflistung 4.23).

```
1 if(isset($_SERVER['QUERY_STRING'])) {
2   setRequestVariable($_SERVER['QUERY_STRING'], &$_GET);
3 }
4 if(isset($argn)) {
5   setRequestVariable($argn, &$_POST);
6 }
```

Auflistung 4.23: Aufruf der Funktion `setRequestVariable()`

Die globale Variable `&_SERVER['DOCUMENT_ROOT']` kann nicht verwendet werden, da diese den Zope-Root enthält und von Zope selbst verwendet wird. WebSIs benötigt aber ebenfalls den Pfad der PHP-Dateien. Hierfür wurde die Variable `&_SERVER['MY_DOCUMENT_ROOT']` eingeführt.

Der Aufruf in Zope erfolgt immer entweder über das „index_html“ Objekt der Startseite, bzw. dem des Memberbereiches. In diesem wird, wie in Auflistung 4.24 angegeben, der PHPGateway aufgerufen.

```
1 <div metal:fill-slot="body">
2   <span tal:replace="structure container/gateway/index.php5" />
3 </div>
```

Auflistung 4.24: Aufruf des PHPGateways in Zope

Das Attribut `metal:fill-slot` bedeutet in Zope dass ein Makro gefüllt wird, indem der angegebene *slot* (hier: body) mit dem Inhalt neu befüllt wird. An der Universität Koblenz ist der *slot* „body“ für den Inhalt innerhalb des Webseiten-Rahmens zuständig. `tal:replace` ersetzt ein Element mit dynamischem Inhalt. Hier wird nun der PHPGateway (ein Zope Objekt, welches den PHP Interpreter aufruft) aufgerufen. `structure` ist hierbei die Angabe, dass eine Struktur, und kein Text (was auch möglich wäre), ersetzt wird.

Für den Aufruf der jeweiligen Aktionen muss jedoch auch der GET-Parameter *action* angegeben werden. Dies geschieht durch das Python-Skript in Auflistung 4.25. Angegeben wird hierbei der Pfad zum „index_html“ Objekt der Hauptseite bzw. des Memberbereiches. An den Pfad angehängt wird nun noch die jeweilige *action*.

```
1 return context.REQUEST.RESPONSE.redirect(
2   "/FB4/Contrib/WebSIs/?action=showCourseOfStudy")
```

Auflistung 4.25: Aufruf einer Aktion in Zope

4.2.2 PHP

Auswertung der Aktionen

Der GET-Parameter *action* wird in der zentralen `index.php5`-Datei verarbeitet. Dazu wird er mit einem Array (siehe Auflistung 4.26) verglichen, der alle möglichen Aktionen enthält.

```
1 $actions = array(
2   "createCourseOfStudy" => array("path" => "courseofstudy",
3                                  "member" => true),
4   "showCourse" => array("path" => "course",
5                         "member" => false)
6   //many more actions
7                 );
```

Auflistung 4.26: Aufbau des Arrays `$actions`

Jede Aktion enthält in diesem Array zwei Informationen, den Pfad und ob es sich um eine Aktion aus dem Memberbereich handelt. Der Pfad besteht jedoch nur aus dem Ordner, in dem die nun aufzurufende, lokale `index.php5`-Datei liegt. Der komplette Pfad zu dieser Datei lässt sich aber anhand dieser beiden Informationen berechnen.

{`$websisRoot`}/{`$infix`}/{`$path`}/`index.php5` (siehe Auflistung 4.27) ist der Pfad zu dieser. `$websisRoot` ist dabei die globale Variable, welche den Wert aus `$_SERVER['MY_DOCUMENT_ROOT']` enthält. `$infix` wird entweder der String „public“ oder „member“, je nachdem ob innerhalb des Arrays `$actions` „`member`“ gleich `true` oder `false` ist, zugewiesen. Doch bevor dieser Pfad aufgerufen werden kann, muss natürlich geprüft werden, ob die angefragte Aktion überhaupt existiert. Dafür wird mit der Funktion `array_key_exists($action, $actions)` überprüft, ob die Aktion vorhanden ist. `$action` ist hier die globale Variable für `&_GET['action']`. Schlägt diese Überprüfung fehl, so wird entweder die Startseite oder der Memberbereich angezeigt, je nachdem in welchem Bereich die Aktion aufgerufen wurde.

```
1 if(!array_key_exists($action, $actions)) {
2   $pageTitle = "Default";
3   $area = "member";
```

```
4     if(stripos($requestUri, "Member") === false) {
5       $area = "public";
6     }
7     include("$websisRoot/$area/index.php5");
8   } else {
9     $path = $actions[$action]["path"];
10    $isMember = $actions[$action]["member"];
11    if($isZope && $isMember
12       && stripos($requestUri,"Member") === false){
13       $redirect .= "Member";
14    } else {
15      $infix = "public";
16      if($isMember) {
17        $infix = "member";
18      }
19      include("{$websisRoot}/{$infix}/{$path}/index.php5");
20    }
21  }
```

Auflistung 4.27: Verarbeitung der Aktion

Auflistung 4.27 enthält weiterhin lediglich noch einige Zeilen, welche benötigt werden um zwischen dem eigenständigen Ausführen (für lokale Tests) oder dem Ausführen innerhalb von Zope zu unterscheiden.

Datenbank

Um die Aktionen verarbeiten zu können benötigt man Zugriff auf die MySQL Datenbank. Dafür wurde vor der Auswertung eine Transaktion gestartet, welche nach dieser beendet wird (siehe Auflistung 4.28).

```
1 include_once("$websisRoot/database/DatabaseConnection.php5");
2 $databaseConnection = new DatabaseConnection("ANONYMOUS");
3 $databaseConnection->beginTransaction();
4
5 //process the action
6
7 $databaseConnection->endTransaction();
8 $databaseConnection->closeConnection();
```

Auflistung 4.28: Starten und Beenden einer Datenbank-Transaktion

Die Datenbankverbindung wird generell als anonymer User aufgebaut, lässt sich aber bei Bedarf ändern. Dafür wird die Methode `changeUserAccount()` verwendet. In Auflistung 4.29 ist diese dargestellt. Der Array `$userAccountData`

enthält dabei wieder ein Array, welches den Benutzernamen, sowie das Passwort für den Account enthält.

```
1  public function changeUserAccount($userAccountIdentifier) {
2    global $websisRoot;
3    include("$websisRoot/database/connectionData.php5");
4    include("$websisRoot/database/userAccounts.php5");
5
6    $dbUser = $userAccountData[$userAccountIdentifier]
7                              ["userAccount"];
8    $dbPassword = $userAccountData[$userAccountIdentifier]
9                                  ["accountPassword"];
10
11   $this->dbConn->change_user($dbUser, $dbPassword, $dbDatabase);
12 }
```

Auflistung 4.29: Ändern des Useraccounts

Neben dem Starten und Beenden von Transaktionen, sowie dem Wechsel des Benutzerkontos wird noch eine Funktion zum Schreiben und Lesen der Daten benötigt. Dafür müssen die in Unterabschnitt 4.1.3 erwähnten gespeicherten Prozeduren aufgerufen werden. Dies geschieht mithilfe der Funktion `executeStoredProcedureCall()` aus Auflistung 4.30.

```
1  public function executeStoredProcedureCall(
2                                        $storedProcedureIdentifier,
3                                        $inParameters=array(),
4                                        $outParameters=array(),
5                                        $inoutParameters=array()){
6    global $output;
7    $result = array();
8
9    $outParameterList = $this->buildParameters("OUT",
10                                              $outParameters);
11   $inoutParameterList = $this->buildParameters("INOUT",
12                                                $inoutParameters);
13   $inParameterList = $this->buildParameters("IN", $inParameters);
14
15   $storedProcedureCall .= "CALL $storedProcedureIdentifier("
16                           . $outParameterList;
17   if(($outParameterList != "" && $inoutParameterList != "")
18      || ($outParameterList != "" && $inParameterList != "")) {
19     $storedProcedureCall .= ",";
20   }
21   $storedProcedureCall .= $inoutParameterList;
22   if($inoutParameterList != "" && $inParameterList != "") {
23     $storedProcedureCall .= ",";
24   }
25   $storedProcedureCall .= $inParameterList;
```

```
26    $storedProcedureCall .= ");";
27
28    $res = $this->dbConn->query($storedProcedureCall);
29
30    if (mysqli_connect_errno()) {
31      $this->abortTransaction();
32      $output .= "Couldn't execute the stored procedure
33                  $storedProcedureIdentifier: %s\n"
34                 . mysqli_connect_error();
35      printf("Couldn't execute the stored procedure
36              $storedProcedureIdentifier: %s\n",
37             mysqli_connect_error());
38      die();
39    }
40
41    $result[0] = $this->saveResult($res);
42    $this->freeStoredProcedureMemory($res);
43    $this->extractOutParametersValue($outParameters, &$result);
44
45    return $result;
46  }
```

Auflistung 4.30: Aufruf von gespeicherten Prozeduren

Die Funktion `buildParameters()` konvertiert die Daten aus PHP in das Format für die gespeicherte Prozedur. Dabei werden auch die Werte der Variablen von ihrer HTML-Darstellung (z.B. unregelmäßig) wieder in ihre Ursprungsform (z.B. unregelmäßig) konvertiert. Danach wird der Aufruf für die gespeicherte Prozedur erstellt und ausgeführt. Sollte dabei ein Fehler aufgetreten sein, so wird die Transaktion abgebrochen und alle Änderungen rückgängig gemacht.

Wenn die gespeicherte Prozedur eine Ausgabe liefert, so muss diese wieder nach PHP und HTML konvertiert werden. Dies geschieht durch die Funktionen `extractOutParametersValue()` (OUT-Parameter) und `saveResult()` (Ergebnismenge).

Das Ergebnis dieser Funktion ist ein Array, dessen Ergebnis sich durch den Schlüssel „0“ abfragen lässt. Fehlercodes und -meldungen lassen sich durch einen vorher angegeben Schlüssel (meist „msgcode“ und „msg“) abrufen.

Verarbeitungssteuerung

Die Steuerung vom Beginn der Verarbeitung einer Aktion, bis zu Ausgabe der Daten wird von einer lokalen `index.php5`-Datei vorgenommen. In Auflistung 4.31 ist die `index.php5`-Datei für den Memberbereich der Studiengänge angegeben. Für den Memberbereich wird zuerst der Login geprüft, dazu wird zuerst geschaut, dass die Variable `$user` (globale Variable für `$_SERVER['ZOPE_USER']`) ungleich „Anonymous User“ ist. Sollte dies nicht der Fall sein, wird man gebeten sich in Zope einzuloggen. Da sich aber jeder Benutzer mit einer Rechnerkennung an der Universität Koblenz über Zope einloggen kann, muss in einem zweiten Schritt geprüft werden, ob der eingeloggte Benutzer die erforderlichen Rechte zum Verwalten der Seite besitzt. Für das Bearbeiten von Studiengängen muss der Benutzer die Rolle des Dekans haben (`if($userManagement->hasRole(DEAN))`).

```
1  //includes, global variables and javascripts are declared here
2
3  if( $user != "Anonymous User" ) {
4    if($userManagement->hasRole(DEAN)) {
5      if ( isset($action) ) {
6        switch ( $action ) {
7          case "createCourseOfStudy": {
8            if ( isset($_POST["cosId_hdn"]) ) {
9              $courseOfStudyFormHandler =
10               new CourseOfStudyFormHandler();
11             $courseOfStudy = $courseOfStudyFormHandler
12                                 ->getCourseOfStudy();
13             if ( $courseOfStudyFormHandler->isCreated() ) {
14               $action = "showMemberCoursesOfStudy";
15               include(
16                   "$websisRoot/member/courseofstudy/index.php5");
17             } else {
18               $courseOfStudyVisualization =
19                 new CourseOfStudyFormVisualization(
20                                                   $courseOfStudy);
21               $pageTitle = "Neuen Studiengang erstellen:";
22               $output .= $courseOfStudyVisualization
23                            ->getCourseOfStudyFormVisualization();
24             }
25           } else {
26             $courseOfStudyVisualization =
27               new CourseOfStudyFormVisualization();
28             $pageTitle = "Neuen Studiengang erstellen:";
29             $output .= $courseOfStudyVisualization
30                          ->getCourseOfStudyFormVisualization();
31           }
32           break;
33         }
34         //many other actions
```

```
35          default: {
36            $messages .= Message::buildInvalidAccessMessage();
37            break;
38          }
39        }
40      }
41    } else {
42      $messages .= Message::buildNoRightsMessage();
43    }
44  } else {
45    $redirect .= "Member/Studiengaenge";
46  }
```

Auflistung 4.31: Die lokale `index.php5`-Datei für den Memberbereich der Studiengänge

Nach einem erfolgreichen Login wird überprüft, welche Aktion ausgeführt werden soll. Ist diese gefunden, so werden die weiteren Schritte eingeleitet. Für die Aktion `createCourseOfStudy` würde zuerst überprüft, ob das Formular gerade abgesendet wurde. Dafür wird die Existenz eines Formularfeldes geprüft. Sollte das Formularfeld existieren, so wird zuerst der `CourseOfStudyHandler` aktiv. Mit dem im Handler erstellten Objekt wird nun bei Erfolg der Aktion die Auflistung der Studiengänge angezeigt. Bei Fehlschlag wird die Klasse `CourseOfStudyVisualizationForm` mit den vorher eingegebenen Daten und der Fehlermeldung aufgerufen.

Sollte die Aktion `createCourseOfStudy` das erste Mal aufgerufen werden (es existieren noch keine Formulardaten), dann wird die Klasse `CourseOfStudyVisualizationForm` direkt und ohne Parameter aufgerufen.

Die Ausgabe wird am Ende der Verarbeitung in der globalen Variable `$output` gespeichert.

Handler

Die Verarbeitung der Formulardaten erfolgt durch die Handler. Auflistung 4.32 zeigt den Konstruktor des Handlers für die Formulardaten der Studiengänge. Dieser liest, wenn die Aktion nicht `moveCourseOfStudyToRecycleBin` ist, die Formulardaten aus. Dabei werden auch Fehler behandelt. Es gibt dabei die Unterscheidung zwischen Fehlern und Warnungen. Bei Warnungen wird einfach ein Standardwert für die jeweilige Variable gespeichert. Dies ist bei leeren Feldern der Fall. Fehler hingegen sind schwerwiegender und es lässt sich nicht einfach ein Standardwert setzten. In Auflistung 4.32 tritt ein Fehler z.B. dann auf, wenn der

Verlust im ersten Jahr keine Zahl größer 0 und kleiner 1 ist. In diesem Fall wird die Variable `$isCreated` auf `false` gesetzt und keine weitere Verarbeitung durchgeführt

```
1  public function CourseOfStudyFormHandler() {
2    global $output, $action, $messages;
3
4    if ($action != "moveCourseOfStudyToRecycleBin") {
5      $lossSecondYear = trim($_POST["lossSecondYear_txt"]);
6      $lossSecondYear = str_replace(",", ".", $lossSecondYear);
7      if (is_numeric($lossSecondYear)
8          && $lossSecondYear >= 0
9          && $lossSecondYear <= 1) {
10       $lossSecondYear = (float) $lossSecondYear;
11     } else if ($lossSecondYear == "") {
12       $lossSecondYear = 0;
13       $messages .= MessageUtility::buildMessage('Keinen Wert für
14                     "Verlust im 2. Jahr" angegeben!
15                     "Verlust im 2. Jahr" wurde auf 0 gesetzt!',
16                     true);
17     } else {
18       $messages .= MessageUtility::buildMessage(
19                     '"Verlust im 2. Jahr" muss eine Zahl
20                      >= 0 und <= 1 sein!', true);
21       $this->isCreated = false;
22     }
23     //many more data is read
24
25     $this->courseOfStudy = new CourseOfStudy(
26                                         $cosId, false,
27                                         $name, $faculty,
28                                         $lossSecondYear,
29                                         $lossThirdYear,
30                                         $freshManStudentCount,
31                                         $moduleGroupsArray,
32                                         $courseTypesArray);
33   }
34
35   if($this->isCreated && $action == "editCourseOfStudy") {
36     $this->isCreated = $this->checkCourseOfStudyDeletionStatus();
37   }
38
39   if ($this->isCreated) {
40     $this->processData();
41   }
42 }
```

Auflistung 4.32: Der Konstruktor für den Handler für die Formulardaten der Studiengänge

Sollte nichts schief gegangen sein, so wird die Methode `processData()` in Auflistung 4.33 aufgerufen. Diese ruft wiederum für die entsprechende Aktion für die Behandlung der Daten auf und speichert deren Rückgabewert in der globalen Variable `$output`.

```
1  private function processData() {
2    global $output, $action, $messages;
3    switch ( $action ) {
4      case "createCourseOfStudy": {
5        $messages .= $this->courseOfStudy
6                              ->storeCourseOfStudy($this->isCreated);
7        break;
8      }
9      //many more cases to handle
10     default: {
11       $messages .= MessageUtility::buildMessage(
12                     "Falscher Parameterwert in
13                     CourseOfStudyFormHandler::
14                     CourseOfStudyFormHandler()!", true);
15       break;
16     }
17   }
18 }
```

Auflistung 4.33: Die Methode `processData()` im Handler für die Formulardaten der Studiengänge

Die Datenklassen

Die Datenklassen enthalten alle für den jeweiligen Bereich nötigen Informationen, bzw. enthalten Methoden um diese aus der Datenbank zu lesen. Ebenso enthalten sie Methoden zum Hinzufügen, Ändern und Entfernen der Daten aus der Datenbank. Die Klasse `CourseOfStudy` in Auflistung 4.34 enthält somit alle Informationen über die Studiengänge. Diese werden in privaten Variablen der Datenklasse gespeichert und können über Getter-Methoden abgerufen werden. Bei seltener verwendeten Informationen, werden diese erst bei einer Anfrage aus der Datenbank ausgelesen, es sei denn, die Daten wurden manuell eingegeben. Dies dient der Aktualität der Daten.

Der Konstruktor bietet zwei Möglichkeiten: das manuelle Erstellen des Objektes, sowie das Auslesen aus der Datenbank. Um zwischen diesen beiden Varianten zu unterscheiden wird der Parameter `$useDatabase` eingeführt. Sollte dieser gleich `true` sein, so werden die Daten aus der Datenbank ausgelesen. Für die häufig verwendeten Informationen geschieht dies direkt in der Methode `extractCourseOfStudyInformation()`. Wenn dies nicht der Fall ist,

dann werden die Daten aus den Konstruktorparametern erstellt und die Variable `$isCreatedManually` auf `true` gesetzt.

Diese Variable dient der Unterscheidung, ob die Daten manuell eingegeben oder aus der Datenbank ausgelesen wurden, und wird für die Getter-Methoden verwendet, welche die Daten bei jedem Zugriff aus der Datenbank auslesen. In der Methode `getCourseTypes()` wird dann nur der derzeitig gespeicherte Wert zurückgegeben, statt die Daten aus der Datenbank auszulesen.

```
1  class CourseOfStudy {
2    //some private variables containing the information
3
4    private $isCreatedManually = false;
5
6    public function CourseOfStudy($id = "", $useDatabase = true,
7                                   $name = "", $faculty = "",
8                                   $lossSecondYear = 0,
9                                   $lossThirdYear = 0,
10                                  $freshManStudentCount = 0,
11                                  $moduleGroupsList = array(),
12                                  $courseTypes = array()) {
13     if ( $useDatabase ) {
14       $this->moduleGroupsList = array();
15       $this->courseTypes = array();
16       $this->id = $id;
17       $this->extractCourseOfStudyInformation();
18     } else {
19       $this->id = $id;
20       $this->name = $name;
21       $this->faculty = $faculty;
22       $this->courseTypes = $courseTypes;
23       $this->lossSecondYear = $lossSecondYear;
24       $this->lossThirdYear = $lossThirdYear;
25       $this->freshManStudentCount = $freshManStudentCount;
26       $this->moduleGroupsList = $moduleGroupsList;
27       $this->isCreatedManually = true;
28     }
29   }
30
31   //the standard Getter-Methods and some special
32   //Getter-Methods like the one below
33
34   public function getCourseTypes() {
35     if ( !$this->isCreatedManually ) {
36       global $databaseConnection;
37       $this->moduleGroupsList = array();
38       $result = $databaseConnection->executeStoredProcedureCall(
39                   "get_all_coursetype_sizes_for_courseofstudy",
40                   array($this->id));
```

```
41        foreach($result[0] as $courseTypeInformation) {
42          $courseType = new CourseType(
43                                  $courseTypeInformation["ctid"],
44                                  $courseTypeInformation["size"]);
45          array_push($this->courseTypes, $courseType);
46        }
47      }
48      return $this->courseTypes;
49    }
50
51    public function deleteCourseOfStudy() {
52      global $databaseConnection;
53      $databaseConnection->changeUserAccount("DEAN");
54      $result = $databaseConnection->executeStoredProcedureCall(
55                  "delete_courseofstudy",
56                  array($_GET["cosid"]), array("msgcode", "msg"));
57      return $result;
58    }
59
60    //many more functions for working on the database
61  }
```

Auflistung 4.34: Die Datenklasse `CourseOfStudy`

Neben der Bereitstellung von Informationen verarbeiten die Datenklassen diese auch. Die Methode `deleteCourseOfStudy()` ist ein Beispiel dafür. Diese Funktion löscht anhand der ID den jeweiligen Studiengang aus der Datenbank. Hierfür wird lediglich die Funktion `executeStoredProcedureCall()` aufgerufen und deren Ergebnis zurückgegeben.

Die Visualisierung

Um die Daten zum Speichern eingeben zu können, muss eine Arbeitsoberfläche geschaffen werden, die dies möglich macht. Dies wird durch die Visualisierungsklassen erreicht. Die Klasse `CourseOfStudyFormVisualization` generiert das Formular zur Eingabe der Daten für die Studiengänge.

Sollte dem Konstruktor (siehe Auflistung 4.35) dabei ein `CourseOfStudy`-Objekt übergeben werden, so werden die Daten aus diesem Objekt in die jeweiligen Formularfelder übertragen. Wenn dies nicht der Fall ist, so wird ein leeres (enthält keine Informationen) `CourseOfStudy`-Objekt erzeugt. Danach wird auch direkt damit begonnen, den HTML-Code für die Ausgabe zu erzeugen.

```
1 public function CourseOfStudyFormVisualization(
2                                       $courseOfStudy=NULL) {
```

```
3     if ($courseOfStudy !== NULL) {
4       $this->courseOfStudy = $courseOfStudy;
5     } else {
6       $this->courseOfStudy = new CourseOfStudy("", false, "", "",
7                                                0, 0, 0, array(),
8                                                array());
9     }
10    $this->buildVisualization();
11  }
```

Auflistung 4.35: Die Konstruktor der Klasse `CourseOfStudyFormVisualization`

Der Code für die Ausgabe wird durch die Funktion `buildVisualization()` in Auflistung 4.36 erzeugt. Dabei wird bei der Aktion beim Absenden des Formulares zwischen dem Erstellen eines neuen und dem Editieren eines alten Studienganges unterschieden. Ansonsten gibt es keine weiteren Unterschiede, zwischen diesen beiden Aktionen, da die Felder beim Erstellen mit den leeren Werten des im Konstruktor erzeugten `CourseOfStudy`-Objekts gefüllt werden.

```
1   private function buildVisualization(){
2     //some variables are declared
3     $this->courseOfStudyFormVisualizationOutput .= "<form
4       action='$_self?{$_SERVER["QUERY_STRING"]}'
5       id='courseOfStudy_form' method='post'>\n";
6     if ($action == "editCourseOfStudy" && isset($_GET["cosid"])) {
7       $courseOfStudyName = $this->courseOfStudy->getName();
8       $replaceHTMLSignsInTitle = false;
9       $pageTitle = "Studiengang \""
10                   . $courseOfStudyName
11                   . "\" editieren:";
12      if($this->courseOfStudy->isDeleted()) {
13        $pageTitle = "";
14        $messages .= MessageUtility::buildMessage(htmlentities(
15                       "Studiengänge, die sich im Papierkorb
16                       befinden, dürfen nicht editiert werden."),
17                       true);
18        $hasError = true;
19      }
20      if($this->courseOfStudy->getName() === NULL) {
21        $pageTitle = "";
22        $messages .= MessageUtility::buildMessage(htmlentities(
23                       "Um einen Studiengang zu editieren, muss
24                       eine gültige Studiengangs-Id angegeben
25                       werden."), true);
26        $hasError = true;
27      }
28      if(isset($_POST["cosoldname"])) {
29        $courseOfStudyName = $_POST["cosoldname"];
```

```
30      }
31      $this->courseOfStudyFormVisualizationOutput .= FormUtility::
32        buildHiddenField("cosoldname", $courseOfStudyName);
33    } else {
34      $messages .= MessageUtility::buildMessage(
35                     "Falscher Parameter oder keine \"cosid\"
36                     in GET angegeben in
37                     CourseOfStudyFormVisualization::
38                     CourseOfStudyFormVisualization()!", true);
39      $hasError = true;
40    }
41
42    if (!$hasError) {
43      //some more code
44      $this->courseOfStudyFormVisualizationOutput .= TableUtility::
45        buildTableBegin("style='width: 100%;'")
46        . TableUtility::buildTableRow(1, array(array(
47          "content" => "Bezeichnung:",
48          "additionalAttributes" => "style='vertical-align:top;
49          font-weight:bold;'"),
50          array("content" => $this->buildNameForm(),
51                "additionalAttributes" => "")))
52              . TableUtility::buildTableRow(2, array(array(
53                "content" => "Fachbereich:",
54                "additionalAttributes" => "style=
55                'vertical-align:top; font-weight:bold;'" ),
56         array("content" => $this->buildFacultyForm(),
57               "additionalAttributes" => "")))
58        //many more table rows
59       . TableUtility::buildTableEnd()
60       . "</form>\n";
61    }
62  }
```

Auflistung 4.36: Die Methode `buildVisualization()`

Die so generierte Ausgabe wird in der lokalen `index.php5`-Datei (siehe Auflistung 4.31) zusammen mit eventuellen Fehlermeldungen des Handlers in der globalen Variable `$output` gespeichert.

Diese wiederum wird in der zentralen `index.php5`-Datei in einen kompletten HTML-Rahmen eingebettet, welcher letztendlich an Zope zurückgegeben und im Browser angezeigt wird.

Die Utilityklassen

Während der Verarbeitung wurde immer wieder auf Utilityklassen zurückgegriffen, diese enthalten häufig verwendete Funktionen. Die Utilityklasse `HeadingUtility` in Auflistung 4.37 dient beispielsweise dem Erstellen von Überschriften. Die Methoden in diesen Klassen sind statisch. `buildHeading()` erstellt eine Überschrift vom Grad
`$headingLevel` mit dem Text `$headingText` im Stil der Universität Koblenz. Durch die Nutzung dieser Hilfsfunktionen, müssen Änderungen am Design nur in dieser Klasse, statt in allen Klassen mit Überschriften gemacht werden.

```
1  class HeadingUtility {
2    public static function buildHeading($headingLevel,
3                                         $headingText) {
4      $heading = "";
5
6      switch ($headingLevel) {
7        case 1: {
8          $heading = "<span class='BodyPageheading'>\n"
9                     . "<p class='BodyPageheading'>$headingText
10                                                        </p>\n"
11                    . "</span>\n"
12                    . "<br />";
13         break;
14       }
15       default: {
16         if($headingLevel > 4) {
17           $headingLevel = 4;
18         }
19         $heading = "<h$headingLevel>$headingText
20                                          </h$headingLevel>\n";
21         break;
22       }
23     }
24     return $heading;
25   }
26 }
```

Auflistung 4.37: Die Utilityklasse `HeadingUtility`

5 Testphase

5.1 Unit- und Systemtest

Nach Abschluss der Implementierungsphase folgt der Übergang in die Testphase. In einem ersten Durchgang wird das gesamte System getestet. Dabei auftretende Fehler und Layoutprobleme, welche im Anschluss behoben werden müssen, werden notiert.

Ein zweiter Testdurchgang dient zur Überprüfung der ersten Fehlerbehebungsphase. Dabei soll festgestellt werden, ob neue Fehler auftreten und ob Fehler übersehen worden sind. Dieser Ablauf wird solange wiederholt, bis keine weiteren (leicht erkennbaren) Fehler auftreten.

5.1.1 Benutzerverwaltung

Entscheidend ist auch die korrekte Umsetzung der Benutzerverwaltung und der Benutzerrechte. Es muss sichergestellt werden, dass die einzelnen Akteure nur auf diejenigen Bereiche des Systems zugreifen können, die ihnen per Definition zugewiesen wurden (s. Abschnitt 2.3). Falls Formulare abgeschickt werden, muss sichergestellt sein, dass ein Benutzer sowohl *vor* deren Aufruf als auch *nach* dem Abschicken der Formulare (noch) über die notwendigen Rechte verfügt.

Dieser Aspekt ist nicht nur, aber speziell im Kontext von Webanwendungen entscheidend, da Webbrowser die Möglichkeit bieten, in der History der Seitenaufrufe zurückzublättern. So kann ein User beispielsweise nach dem Abschicken eines Formulars erneut das Formular aufrufen, indem er einfach auf den „Zurück“-Button klickt. Es kann nun der Fall auftreten, dass diesem User in der Zwischenzeit die notwendigen Rechte zum Aufrufen und Abschicken des Formulars entzogen wurden oder sich der User aus dem System ausgeloggt hat. In diesen Fällen ist es demnach entscheidend, dass auf Serverseite nach dem Abschicken von Formulardaten erneut überprüft wird, ob der aktuelle User (noch) über die notwendigen Rechte verfügt.

5.2 Betatest

Ein Betatest ist ein Systemtest, bei dem unter den Systemvoraussetzungen und mit den Originaldaten getestet werden, welche letztlich im anschließenden praktischen Betrieb vorherrschen und zum Einsatz kommen. Dabei wird die gesamte Funktionalität des Systems abgedeckt.

Damit der Betatest systematisiert ablaufen kann, müssen zunächst die relevanten Systemakteure identifiziert werden. Das UseCase-Diagramm in Abbildung 2.1 enthält alle notwendigen Informationen, so dass folgende Akteure unterschieden werden können:

- Anonymer User,
- Modulverantwortlicher,
- Studiendekan und
- Stundenplanersteller.

Die Rolle des Administrators kann dabei ignoriert werden, da dessen Rechte an den Funktionalitäten, welche die Webinterfaces bieten, durch die restlichen Rollen abgedeckt werden und sich seine zusätzlichen Rechte auf den direkten Zugriff auf die Datenbank beziehen.

5.2.1 Anonymer User

Ein Anonymer User testet dabei folgende Bereiche:

- Übersicht der Studiengänge (s. Unterabschnitt 6.1.2),
- das Modulhandbuch (Übersicht der Module und Lehrveranstaltungen, s. Unterabschnitt 6.1.3),
- das Planungssheet und die Übersicht von Lehrveranstaltungsinstanzen (s. Unterabschnitt 6.1.4) und
- die Übersicht der Qualifikationsziele (s. Unterabschnitt 6.1.5).

Die Rolle des Anonymen Users im Betatest übernehmen Sabine Orth (orth@uni-koblenz.de), Sascha Rutenbeck (srutenbeck@uni-koblenz.de), Dr. Volker Riediger

(riediger@uni-koblenz.de) und Dr. Thorsten Gipp (tgi@uni-koblenz.de).

5.2.2 Modulverantwortlicher

Ein Modulverantwortlicher testet:

- das Editieren von eigenen Module und das Verschieben eigener Module in den Papierkorb (s. Abschnitt 6.1.3),
- das Editieren eigener Lehrveranstaltungen und das Verschieben eigener Lehrveranstaltungen in den Papierkorb. (s. Abschnitt 6.1.3)

Die Rolle des Modulverantwortlichen im Betatest übernehmen Prof. Dr. Jürgen Ebert (ebert@uni-koblenz.de).

5.2.3 Studiendekan

Der Studiendekan testet:

- das Erstellen, Editieren und Verschieben in den Papierkorb von Studiengängen (s. Unterabschnitt 6.1.2),
- das Erstellen, Editieren und Verschieben in den Papierkorb von Modulgruppen (s. Unterabschnitt 6.1.6),
- das Erstellen von Modulen (s. Unterabschnitt 6.1.7),
- das Erstellen von Lehrveranstaltungen (s. Unterabschnitt 6.1.8),
- das Erstellen, Editieren und Verschieben in den Papierkorb von Personen im Personalbestand (s. Unterabschnitt 6.1.9),
- das Erstellen, Editieren und Löschen von Qualifikationszielen (s. Unterabschnitt 6.1.5),
- die Verwaltung des Papierkorbs (Löschen und Wiederherstellen, s. Unterabschnitt 6.1.10),
- und die Verwaltung der Verifikation (Bestätigen und Ablehnen von Änderungen an Modulen und Lehrveranstaltungen, s. Unterabschnitt 6.1.11).

Die Rolle des Studiendekans im Betatest übernehmen Prof. Dr. Jürgen Ebert (ebert@uni-koblenz.de).

5.2.4 Stundenplanersteller

Der Stundenplanersteller testet:

- das Erstellen, Editieren und Löschen von Lehrveranstaltungsinstanzen (s. Abschnitt 6.1.3).

5.3 Bugzilla

Während der Betatestphase werden die auftretenden Fehler, Layoutprobleme und eventuelle Änderungswünsche zentral eingetragen, verwaltet und gepflegt. Zu diesem Zweck benötigt man sog. Modification Request Control Systems (MRCSs) oder Problem Monitoring Systems (PMS). Für diese Studienarbeit wird das Tool *Bugzilla*[1] verwendet, welches zur Verwaltung und Verfolgung von Fehlerberichten (Bugreports) dient.

Unter http://serres.uni-koblenz.de/bugzilla wurde zu diesem Zweck ein „Product“ n „WebSIs“angelegt, so dass hier die Bugreports von den Betatestern eingetragen werden können.

5.4 Browsertypen

Webanwendungen müssen mit verschiedenen Browsertypen getestet werden. Zum Einen kann es bei verschiedenen Browsertypen zu unterschiedlichen, teilweise unschönen Layoutdarstellungen kommen. Zum Anderen unterscheiden sich die in den verschiedenen Browsern eingesetzen JavaScript-Interpreter derart, so dass JavaScript-Anweisungen, welche z.B. im Firefox-Browser einwandfrei funktionieren, im Internet Explorer anders oder gar nicht ausgeführt werden. Aber auch ältere Versionen des selben Browsers können zu diesem Problem führen.

Aus genannten Gründen wird das System mit dem Internet Explorer (ab Version 6.0), Firefox (ab Version 2.0), Opera (ab Version 9.10), Netscape (ab Version 7.1) und Safari (ab Version 3.0.3) getestet.

[1] `http://www.bugzilla.org/` (abgerufen am 30.11.2007)

Mit dem Internet Explorer testen Sabine Orth.

Mit dem Firefox-Browser testen Prof. Dr. Jürgen Ebert und Sascha Rutenbeck.

Mit dem Opera-Browser testen Sabine Orth.

Mit dem Safari-Browser testen Dr. Volker Riediger.

6 Benutzerhandbuch

6.1 Die Webseite

Die WebSIs-Seiten in Zope (siehe Abbildung 6.1) sind so aufgebaut, dass sie an der linken Seite ein Navigationsmenü mit den verfügbaren Aktionen bieten. Das Menü enthält öffentliche Funktionen, sowie einen Memberbereich, für den man sich einloggen muss. Einloggen kann sich jeder, der an der Universität Koblenz eine Rechnerkennung besitzt. Jedoch nur die Personen, welche die entsprechenden Rollen besitzen (siehe Abschnitt 2.3) bekommen auf diesen Seiten Informationen angezeigt. Zum Navigieren befindet sich auf vielen Seiten auch ein Link zur vorherigen Seite direkt unter der blauen Überschrift.

Die folgenden Unterabschnitte tragen die selben Namen, wie die entsprechenden Aktionen im Navigationsmenü.

Abbildung 6.1: WebSIs-Startbildschirm

6.1.1 Die Symbole

Beim Browsen durch die Webseite trifft man immer wieder auf verschiedene Symbole. Deren Bedeutung wird in Tabelle 6.1 erläutert:

	*Editieren*	Editiert den jeweiligen Eintrag.
	*Lehrveranstaltungsinstanz erstellen*	Eine Instanz der jeweiligen Lehrveranstaltung wird erstellt.
	*Löschen*	Der jeweilige Eintrag wird unwiderruflich gelöscht.
	*In den Papierkorb verschieben*	Der jeweilige Eintrag wird in den Papierkorb verschoben und kann vom Dekan wiederhergestellt werden.
	*Aus dem Papierkorb wiederherstellen*	Stellt den gewählten Eintrag aus dem Papierkorb wieder her.
	*E-Mail schreiben*	Über diesen Link lassen sich E-Mails an die entsprechende Person schrieben.
	*Whitepages anzeigen*	Die Whitepages, mit Näheren Informationen zur jeweiligen Person, werden angezeigt.

Tabelle 6.1: Die Symbole in WebSIs

6.1.2 Studiengänge

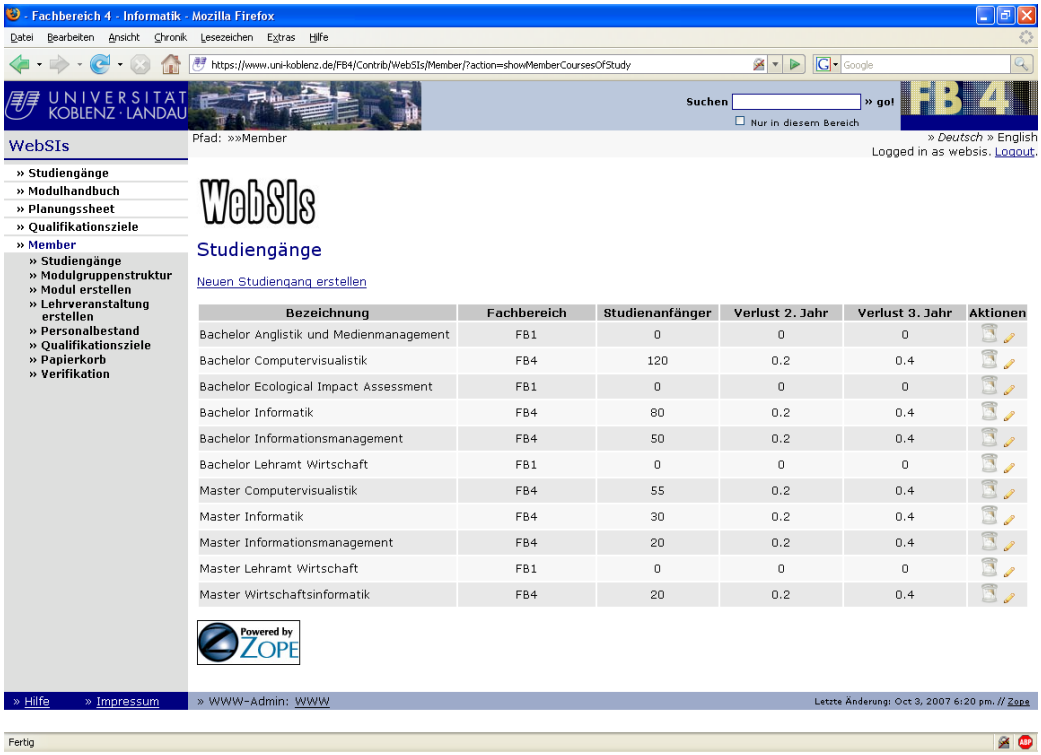

Abbildung 6.2: Studiengänge

Dieser Bereich (siehe Abbildung 6.2) zeigt alle Studiengänge mit zusätzlichen Informationen. Für den öffentlichen Bereich sind diese lediglich die Bezeichnung des Studiengangs, sowie der zugehörige Fachbereich.

Für die im Memberbereich angezeigte Seite existieren zusätzliche Informationen sowie verschiedene Aktionen um die Studiengänge zu bearbeiten. Zum Erstellen neuer Studiengänge befindet sich der Link „Neuen Studiengang erstellen“ direkt unter der blauen Überschrift.

Studiengang erstellen/bearbeiten

Möchte man einen Studiengang erstellen/bearbeiten, so gelangt man mit einem Klick auf die entsprechende Aktion in der Maske aus Abbildung 6.2 zum Eingabe-Formular (siehe Abbildung 6.3).In diesem Formular lässt sich ein Studiengang erstellen, bzw. bearbeiten.

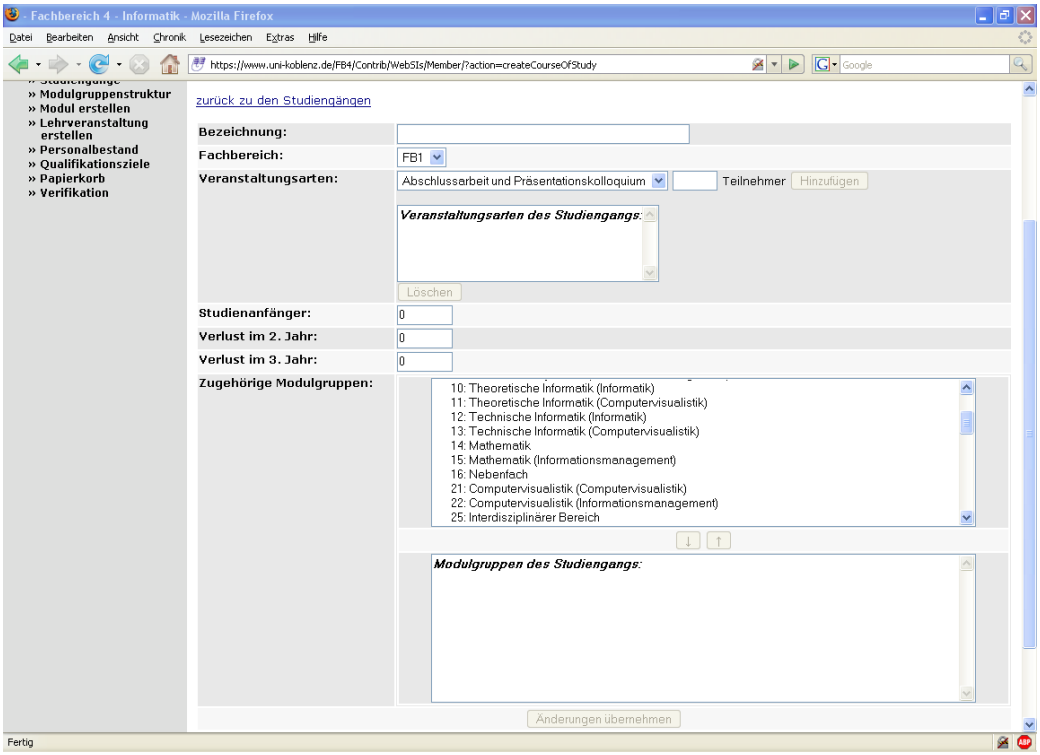

Abbildung 6.3: Studiengänge erstellen/bearbeiten

Die Bezeichnung ist bei der Eingabe der Daten Pflicht. Erst nachdem diese eingegeben wurde, wird der Sende-Button aktiv.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

Veranstaltungsarten — Die Teilnehmerzahl muss eine positive ganze Zahl sein. Wurde keine Zahl angegeben, so wird dieser Wert auf 0 gesetzt.

Studienanfänger — Die Zahl für die Studienanfänger muss eine positive ganze Zahl sein.

Verlust im 2./3. Jahr — Der Verlust (gemeint ist die Rate der Studierenden, welche ihr Studium im entsprechenden Jahr abbrechen) im 2./3. Jahr muss eine Zahl ≥ 0 und ≤ 1 sein. Diese kann mit einem „,“ oder einem „.“ dargestellt werden. Wurde keine Zahl angegeben, so wird dieser Wert auf 0 gesetzt.

6.1.3 Modulhandbuch

Abbildung 6.4: Modulhandbuch

Dieser Link zeigt das Modulhandbuch aus Abbildung 6.4 an. Standardmäßig wird beim Laden das Modulhandbuch für alle Studiengänge angezeigt. Es lässt sich über eine Dropdown-Box aber auch nur ein spezieller Studiengang anzeigen.

Angezeigt werden für diesen Studiengang die Modulgruppen, sowie ihre Submodulgruppen. Ab der Ebene der Module sind die Einträge mit einem Link versehen. Mit einem Klick auf diesen erhält man weitere Informationen zur jeweiligen Veranstaltung.

Veranstaltungsinformationen

Hat man beim Modulhandbuch ein Modul oder eine Lehrveranstaltung ausgewählt, so gelangt man zu den Veranstaltungsinformationen (siehe Abbildung 6.5). Dort werden alle Informationen zu dieser Veranstaltung angezeigt. Ebenfalls befinden sich dort weitere Aktionen, welche mit den jeweiligen Rechten ausgeführt werden können.

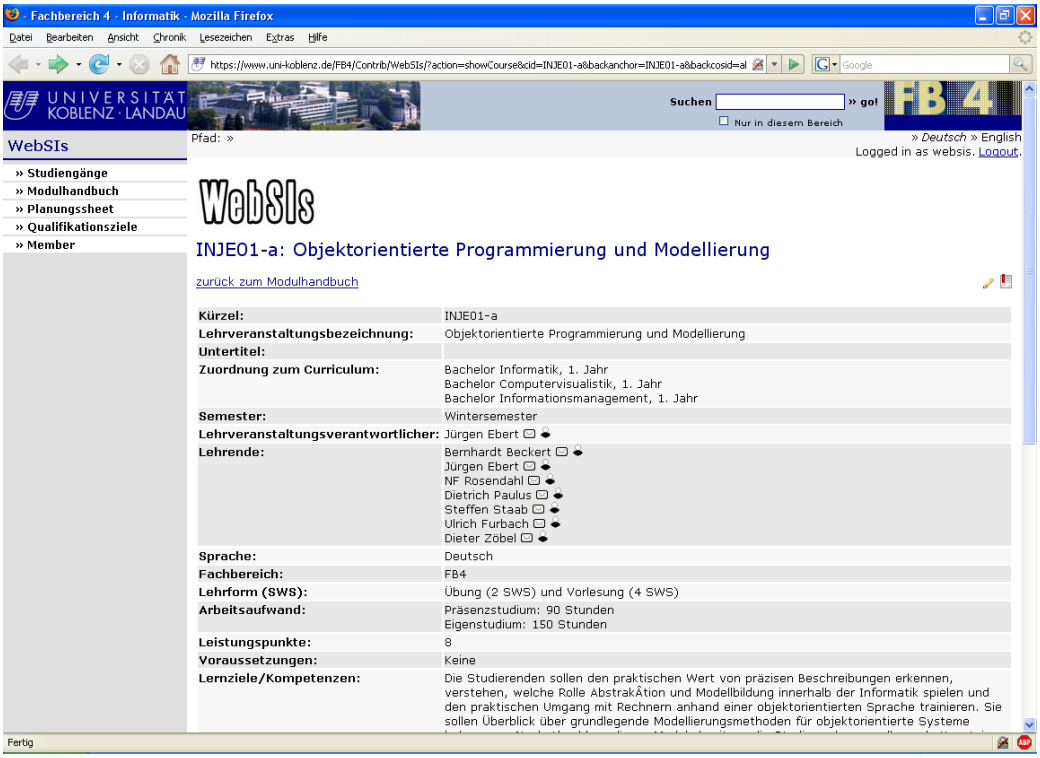

Abbildung 6.5: Veranstaltungsinformationen

Lehrveranstaltungsinstanz erstellen/bearbeiten

Eine der Aktionen, welche sich bei den Veranstaltungsinformationen auswählen lassen, ist das Erstellen einer Lehrveranstaltungsinstanz (Instanz dieser Veranstaltung für genau ein Semester). Dort (siehe Abbildung 6.6) lässt sich eine Lehrveranstaltungsinstanz erstellen bzw. bearbeiten. Die erstellten Lehrveranstaltungsinstanzen werden dann im Planungssheet (siehe Unterabschnitt 6.1.4) angezeigt.

Bei den Lehrveranstaltungsinstanzen ist die Angabe von „Semester und Jahr“ sowie von einer URL notwendig. Erst danach aktiviert sich der Sende-Button.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

Semester und Jahr — Das Format des Jahres muss im Format YYYY (für Sommersemester) bzw. YYYY/YYY(Y+1) (für das Wintersemester) angegeben werden.

URL — Die URL sollte eine gültige Form aufweisen (z.B. `http://www.uni-koblenz.de/~brchrist`).

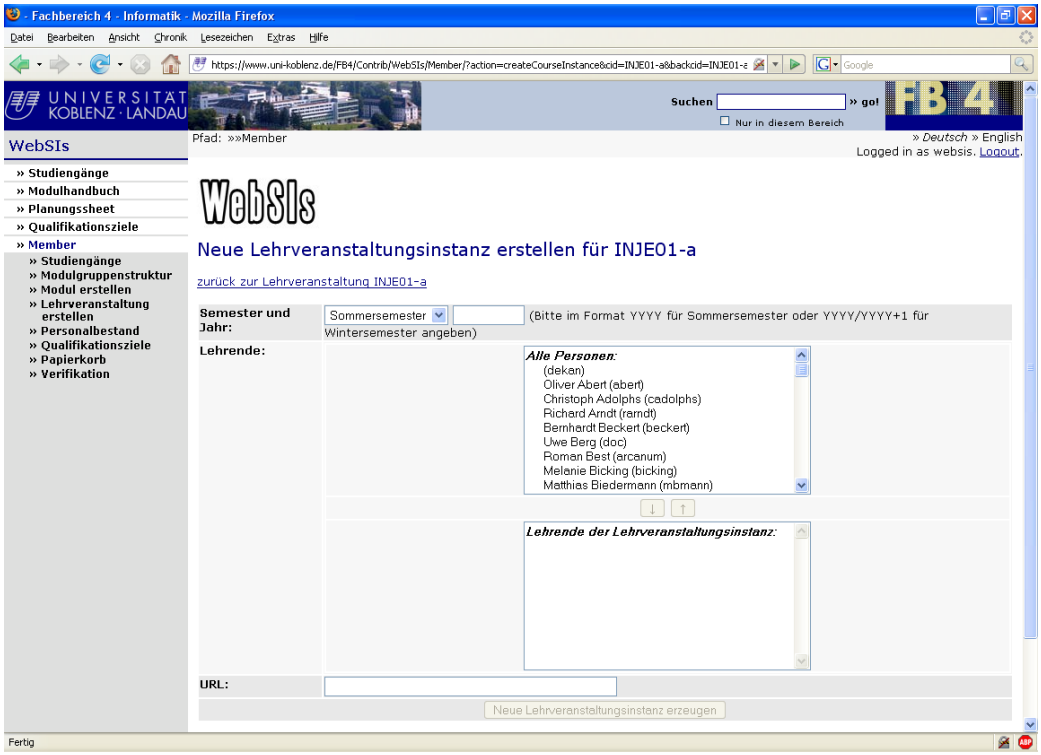

Abbildung 6.6: Lehrveranstalungsintanz

Lehrveranstaltung bearbeiten

Bei den Lehrveranstaltungsinformationen gibt es eine Aktion für deren Bearbeitung. Weitere Informationen hierzu befinden sich in Unterabschnitt 6.1.8.

Modul bearbeiten

Eine weitere Aktion, jedoch nur für Module, ist das Bearbeiten von Modulen. Weitere Informationen hierzu befinden sich in Unterabschnitt 6.1.7.

6.1.4 Planungssheet

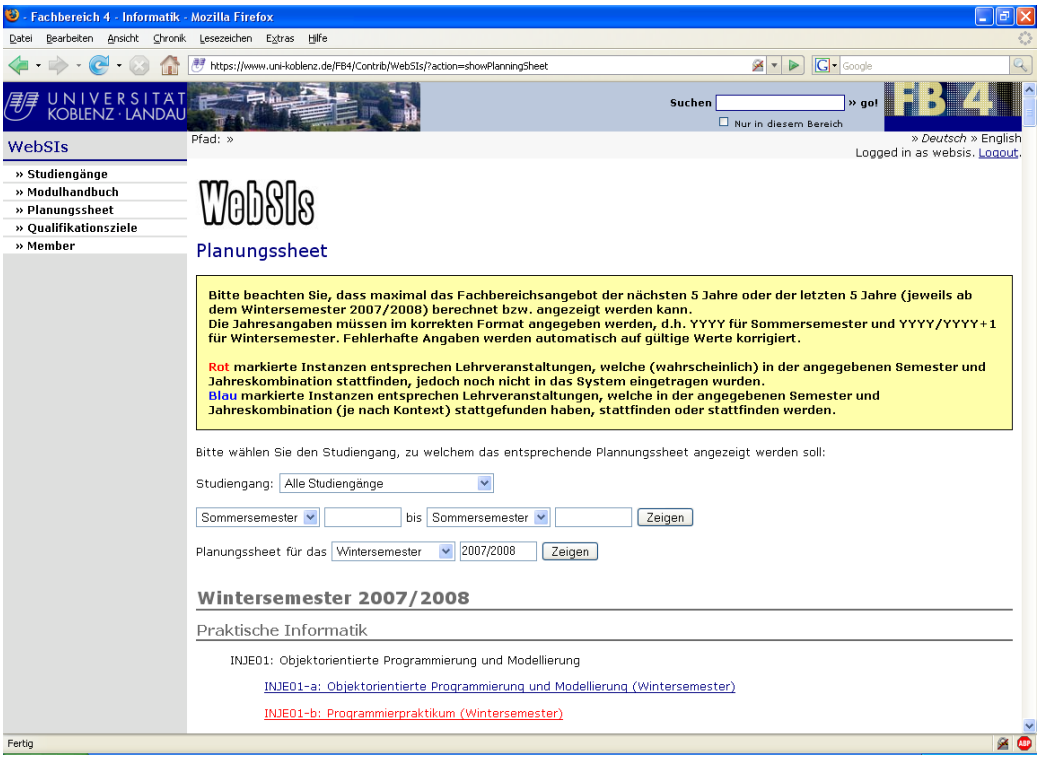

Abbildung 6.7: Planungssheet

Das Planungssheet (siehe Abbildung 6.7) liefert eine Übersicht über alle Lehrveranstaltungen, die in der Vergangenheit angeboten wurden, aktuell stattfinden und in Zukunft angeboten werden. Die Ansicht ist ähnlich der des Modulhandbuchs (siehe Unterabschnitt 6.1.3).

Man kann sich das Planungssheet sowohl für nur einen Studiengang anzeigen lassen, als auch den Zeitraum bestimmen, für den es angezeigt werden soll.

Diesmal sind nur die Lehrveranstaltungen verlinkt. Sind die Links rot, so handelt es sich um eine noch nicht erstellte Lehrveranstaltungsinstanz und man gelangt mit einem Klick auf diese (und einem erfolgreichen Login) zum Eingabe-Formular (siehe Abschnitt 6.1.3). Bei den blauen Links wird eine Seite angezeigt, die Informationen zu der Lehrveranstaltung und weitere Informationen der Instanz enthält (ähnlich Abbildung 6.5).

6.1.5 Qualifikationsziele

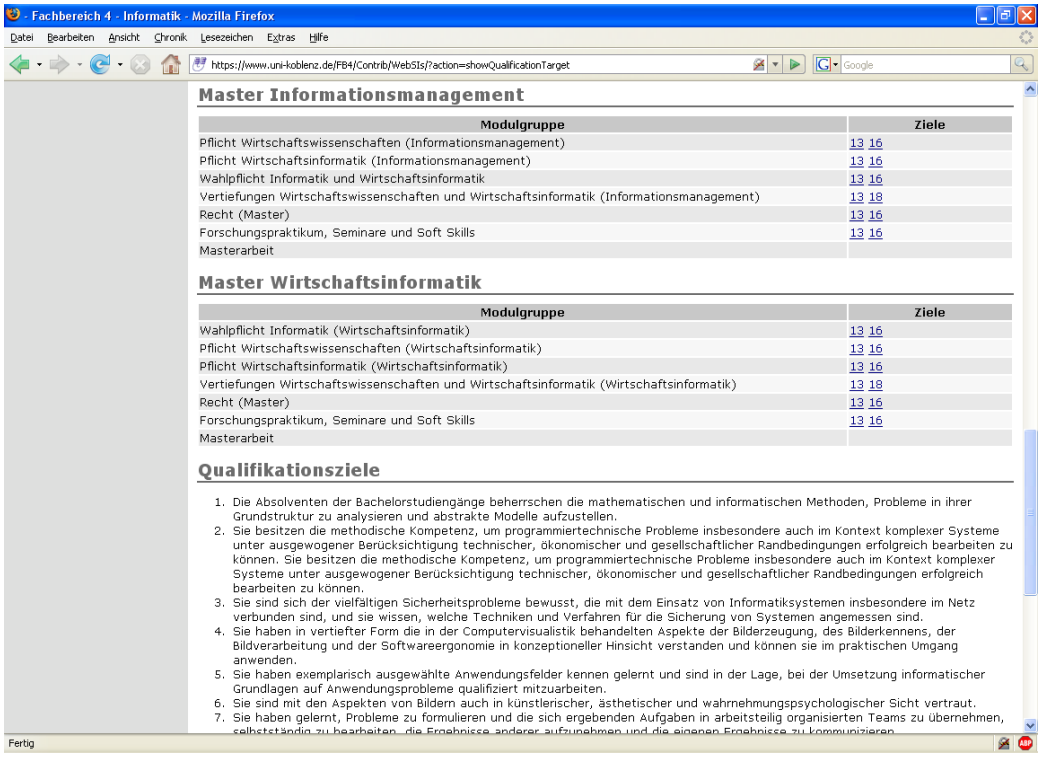

Abbildung 6.8: Qualifikationsziele

Standardmäßig werden hier (siehe Abbildung 6.8) die Qualifikationsziele für jeden Studiengang angezeigt. Man kann aber auch über die Dropdown-Box nur einen Studiengang auswählen. Neben den Modulgruppen eines Studiengangs sind dessen Ziele aufgelistet. Mit einem Klick auf eine Nummer gelangt man zum Text des jeweiligen Qualifikationsziels.

Qualifikationsziele erstellen/bearbeiten

Im Memberbereich findet sich unter „Qualifikationsziele“ eine Übersicht aller Qualifikationsziele, sowie ein Formularfeld um neue Qualifikationsziele zu erstellen. Abbildung 6.9 zeigt dies.

Beim Klick auf Bearbeiten wird der zu bearbeitende Text in das Formularfeld geladen und lässt sich bearbeiten.

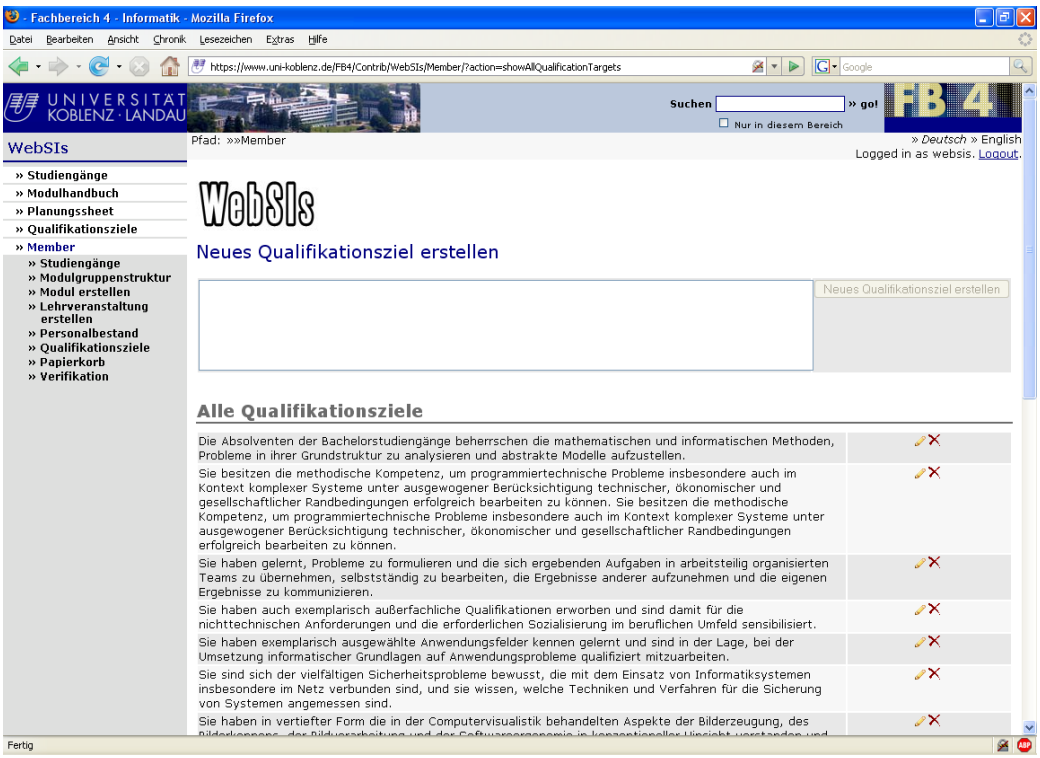

Abbildung 6.9: Qualifikationsziele erstellen/bearbeiten

6.1.6 Modulgruppenstruktur

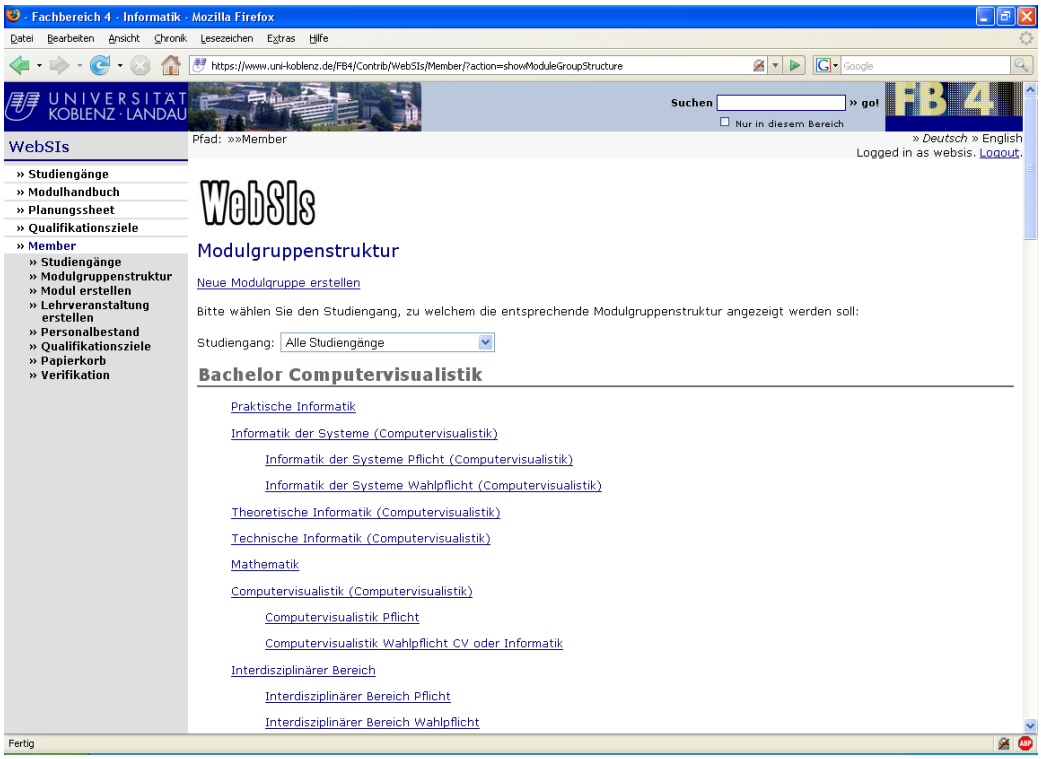

Abbildung 6.10: Modulgruppenstruktur

Die Modulgruppenstruktur zeigt die Struktur der Modulgruppen und deren Submodulgruppen an (siehe Abbildung 6.10. Diese wird standardmäßig für alle Studiengänge angezeigt, lässt sich aber auf einen Studiengang begrenzen.

Mit einem Klick auf eine Modulgruppe lässt sich diese bearbeiten bzw. über den Link „Neue Modulgruppe erstellen“ eine neue erstellen.

Modulgruppe erstellen/bearbeiten

Hat man einen der beiden Links ausgewählt, so gelangt man in die Maske aus Abbildung 6.11. Dort lässt sich eine Modulgruppe erstellen bzw. bearbeiten.

Pflicht ist es einen Namen für die Modulgruppe anzugeben. Erst danach wird der Sende-Button aktiv.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

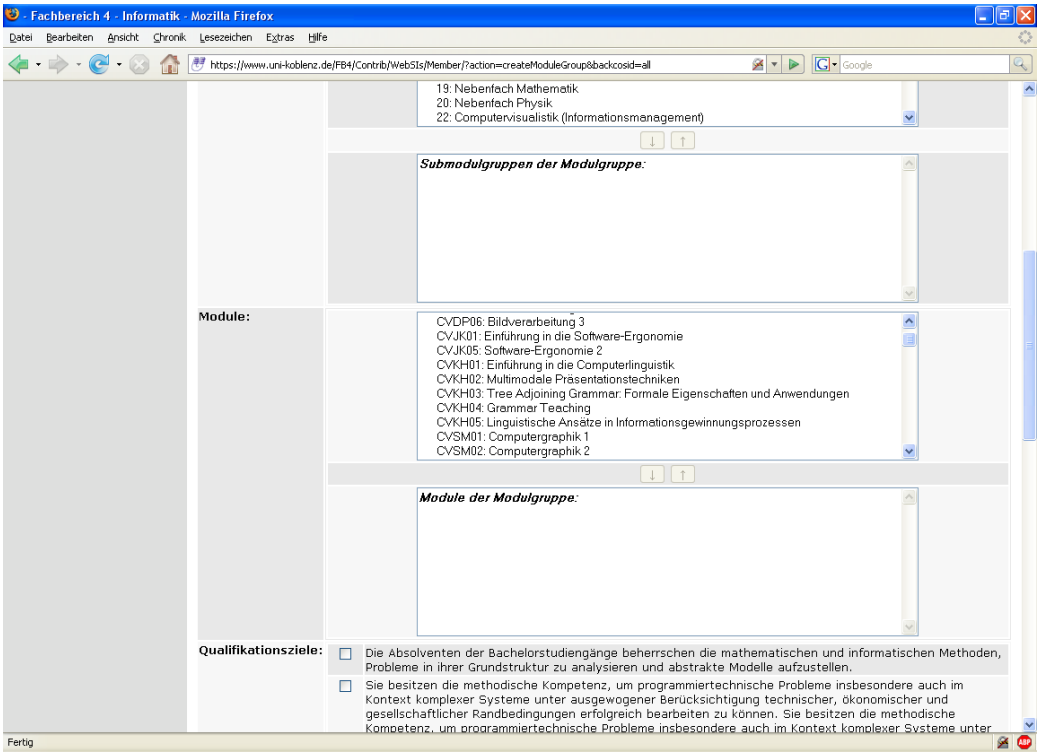

Abbildung 6.11: Modulgruppenstruktur erstellen/bearbeiten

Leistungspunkte — Die Leistungspunkte müssen eine ganze, positive Zahl sein. Wurde kein Wert eingegeben, so wird dieser auf 0 gesetzt.

Submodulgruppen — Submodulgruppen können nur angegeben werden, wenn noch keine Module angegeben wurden.

Module — Module können nur angegeben werden, wenn noch keine Submodulgruppen angegeben wurden.

6.1.7 Modul erstellen

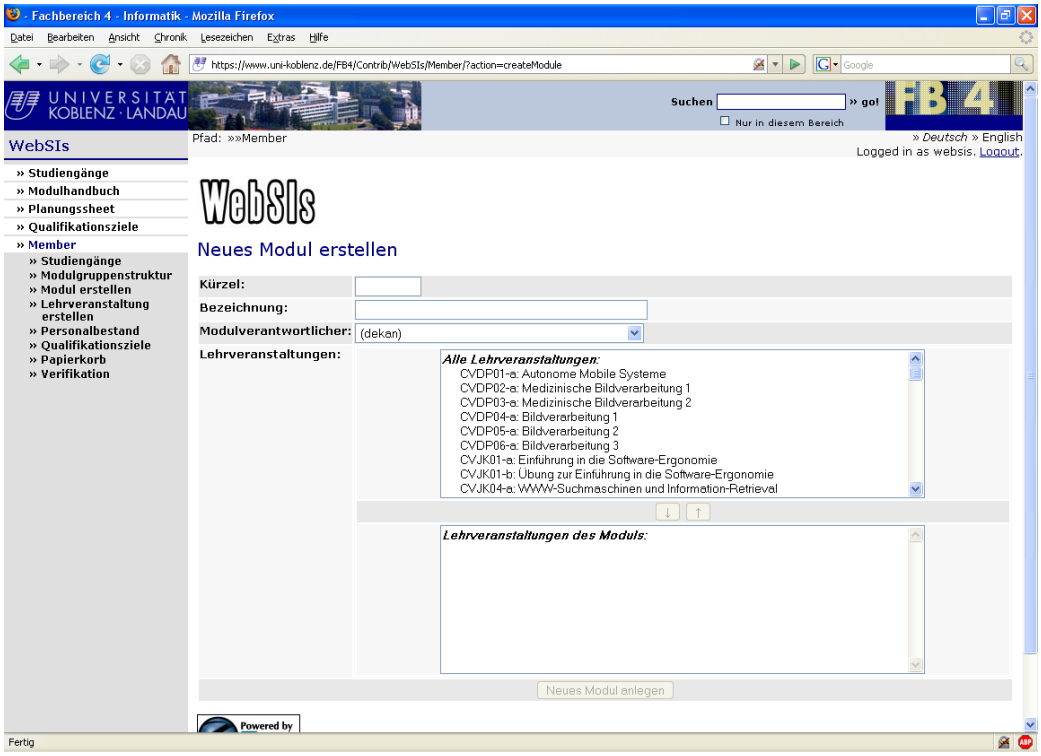

Abbildung 6.12: Modul erstellen

Dieser Menüpunkt erstellt ein neues Modul. Die Maske (siehe Abbildung 6.12) ist die selbe wie für das Bearbeiten von Modulen (siehe Unterabschnitt 6.1.3).

Wurde ein Modul von dem Modulverantwortlichen bearbeitet, so müssen die Änderungen noch vom Studiendekan verifiziert werden.

Die Eingabe eines Kürzels ist für Module Pflicht um den Sende-Button zu aktivieren.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

Kürzel — Das Kürzel muss genau 6 Zeichen lang sein und darf keine Leerzeichen enthalten. Generell hat es die Form SSLLNN (SS = Studiengangskürzel; LL = Kürzel des Lehrenden; NN = durchnummerierte Ziffer bezüglich SSLL).

6.1.8 Lehrveranstaltung erstellen

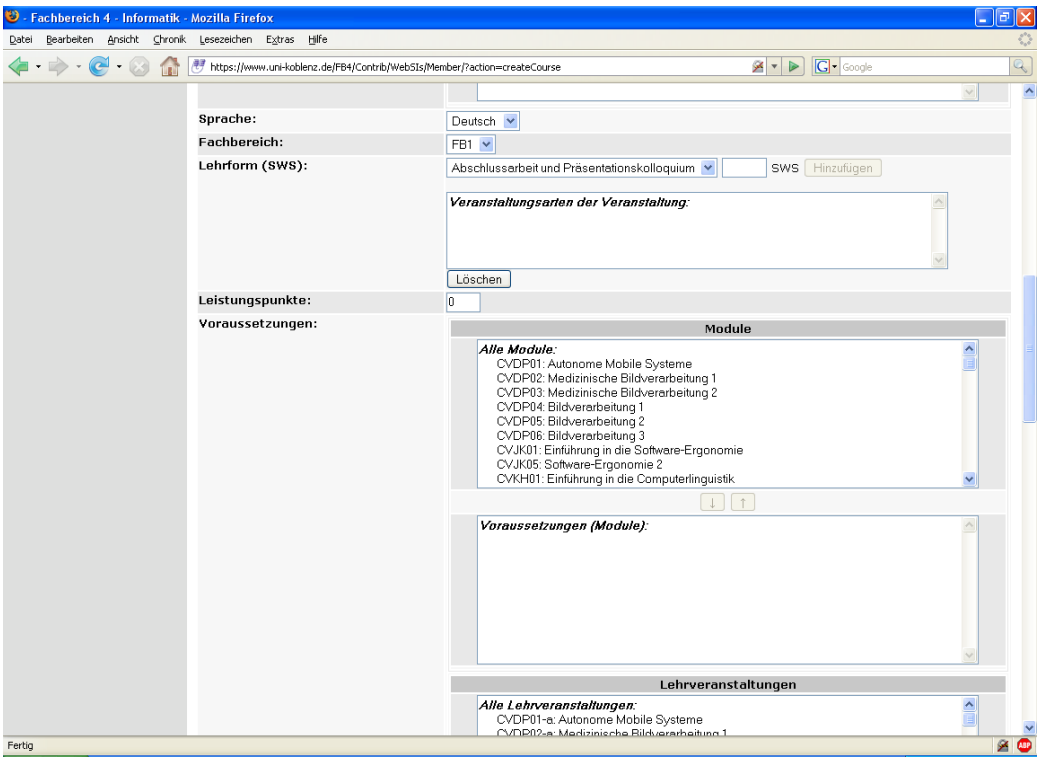

Abbildung 6.13: Lehrveranstaltung erstellen

Dieser Menüpunkt erstellt eine neue Lehrveranstaltung. Die Maske (siehe Abbildung 6.13) ist die selbe wie für das Bearbeiten von Lehrveranstaltungen (siehe Unterabschnitt 6.1.3).

Wurde ein Modul von dem Modulverantwortlichen bearbeitet, so müssen die Änderungen noch vom Studiendekan verifiziert werden.

Die Eingabe eines Kürzels ist für Lehrveranstaltungen Pflicht um den Sende-Button zu aktivieren.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

Kürzel — Das Kürzel muss genau 8 Zeichen lang sein und darf keine Leerzeichen enthalten. Generell hat es die Form SSLLNN-b (SS = Studiengangskürzel; LL = Kürzel des Lehrenden; NN = durchnummerierte Ziffer bezüglich SSLL; b = durchnummerierter Buchstabe bezüglich SSLLNN).

Faktor — Der Faktor muss eine Zahl ≥ 0 und ≤ 1 sein. Diese kann mit einem

„,“ oder einem „.“ dargestellt werden.

Zuordnung zum Curriculum — Die Jahresangabe muss eine ganze, positive Zahl sein. Gemeint ist die Angabe des Studienjahres, in welchem der Student diese Veranstaltung hören sollte.

Leistungspunkte — Die Leistungspunkte müssen eine ganze, positive Zahl sein. Wurde dieser Wert nicht eingegeben, so wird er auf 0 gesetzt.

Literatur — Die „BibTeX-Keys der LitDB“ dürfen keine Leerzeichen enthalten.

6.1.9 Personalbestand

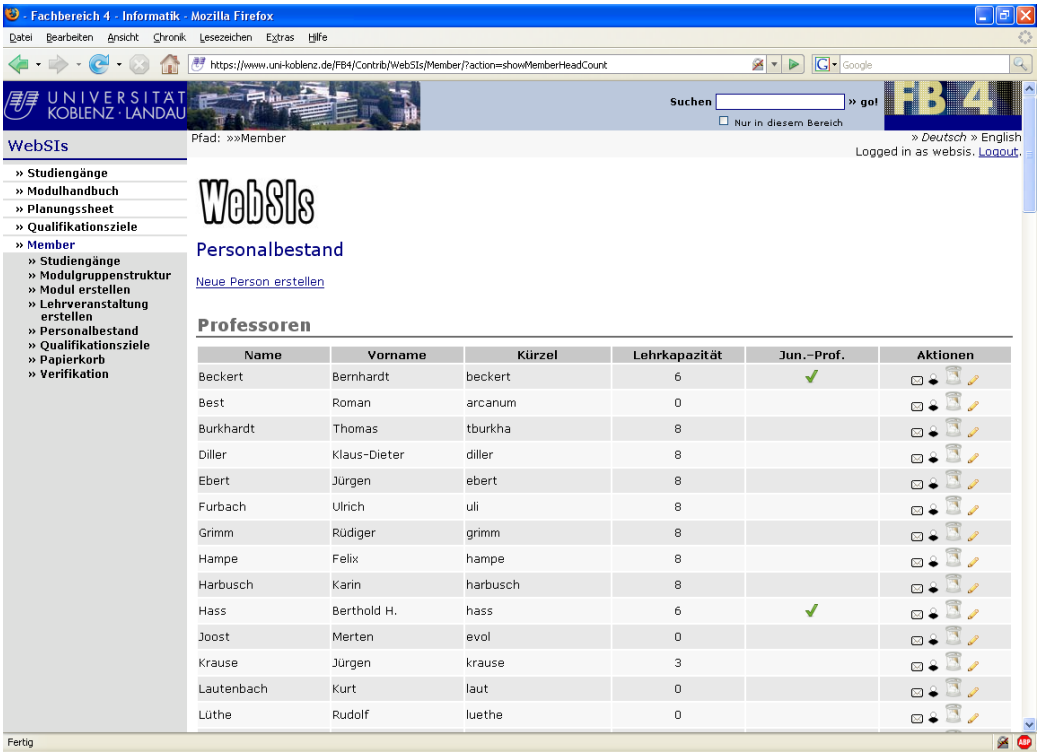

Abbildung 6.14: Personalbestand

Diese Seite (siehe Abbildung 6.14) zeigt den Personalbestand an. Hier lassen sich auch neue Personen erstellen, in den Papierkorb verschieben und bearbeiten.

Person erstellen/bearbeiten

Beim Klick auf den Link zum Erstellen oder Bearbeiten einer Person öffnet sich das Eingabe-Formular aus Abbildung 6.15. Diese Maske dient dem Erstellen und Bearbeiten von Personen.

Die Eingabe eines Kürzels ist für Personen Pflicht um den Sende-Button zu aktivieren.

Bei der Eingabe der Daten sollte Folgendes beachtet werden:

Kürzel — Das Kürzel darf keine Leerzeichen enthalten.

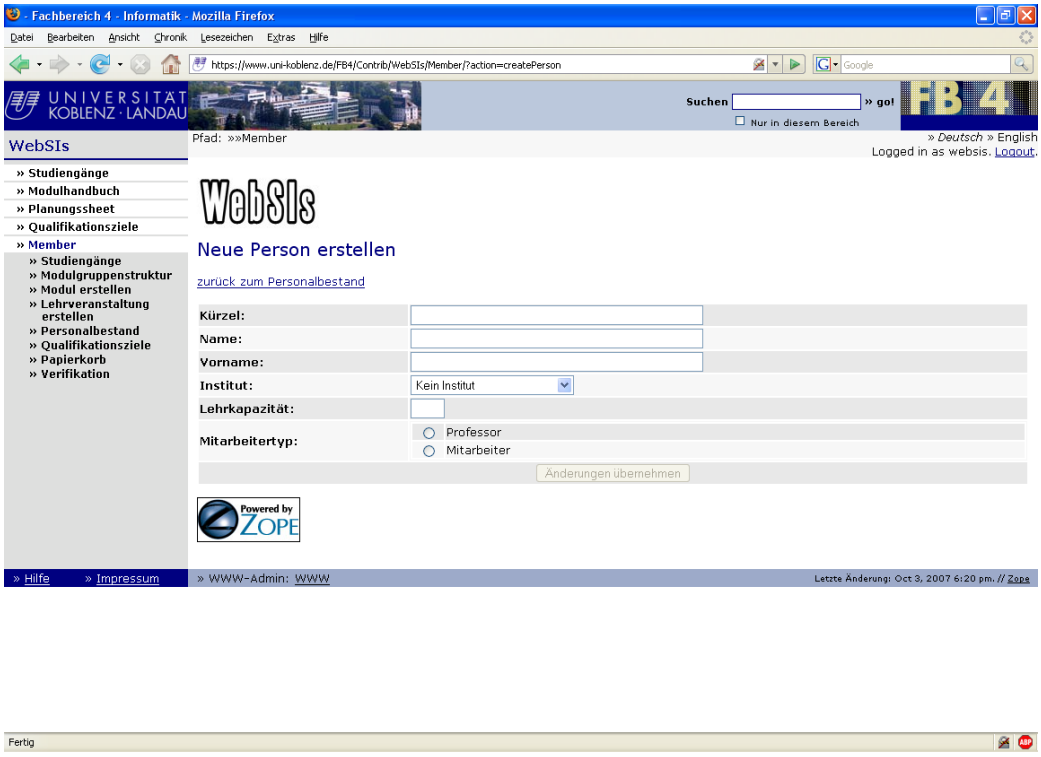

Abbildung 6.15: Person erstellen/bearbeiten

Lehrkapazität — Die Lehrkapazität muss eine ganze, positive Zahl sein. Wurde kein Wert eingegeben, so wird dieser auf 0 gesetzt.

Mitarbeitertyp — Wenn dieser Wert nicht angegeben wird, so wird er auf „Mitarbeiter“ gesetzt.

6.1.10 Papierkorb

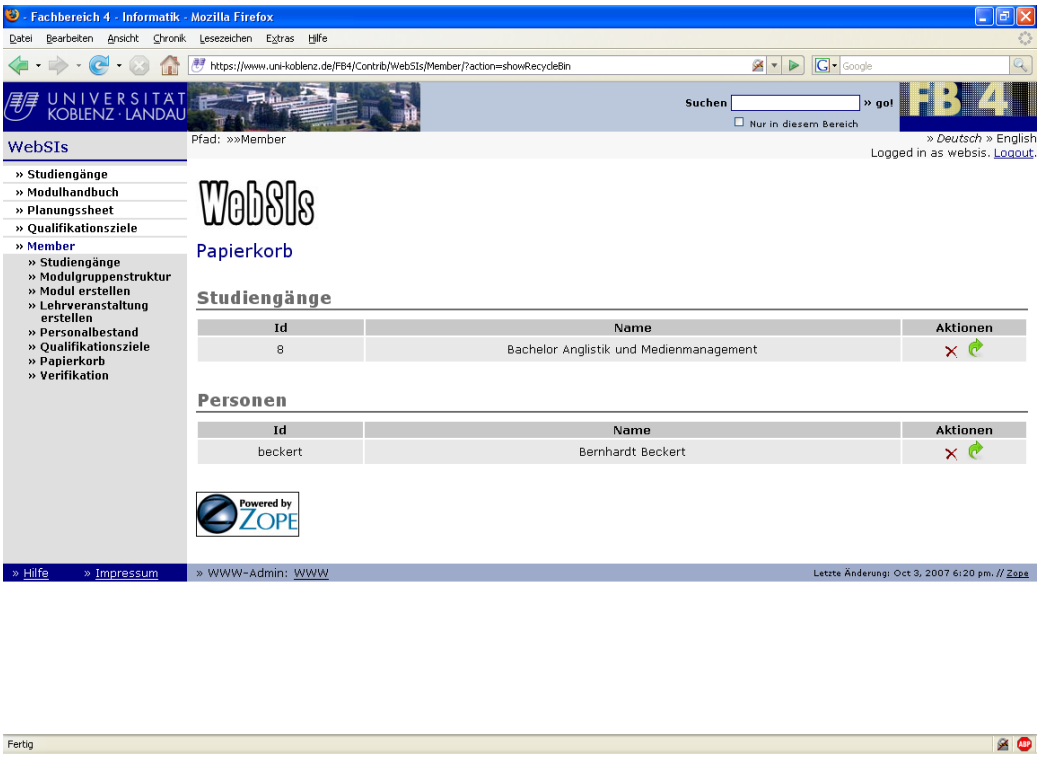

Abbildung 6.16: Papierkorb

Der Papierkorb (siehe Abbildung 6.16) zeigt alle in diesen verschobenen Objekte an. Dies können Studiengänge, Modulgruppen, Module, Lehrveranstaltungen und Personen sein. Man hat die Möglichkeit diese hier entweder vollständig zu löschen oder wiederherzustellen. Wenn weitere Informationen zu einem Objekt existieren, so ist der Name verlinkt.

6.1.11 Verifikation

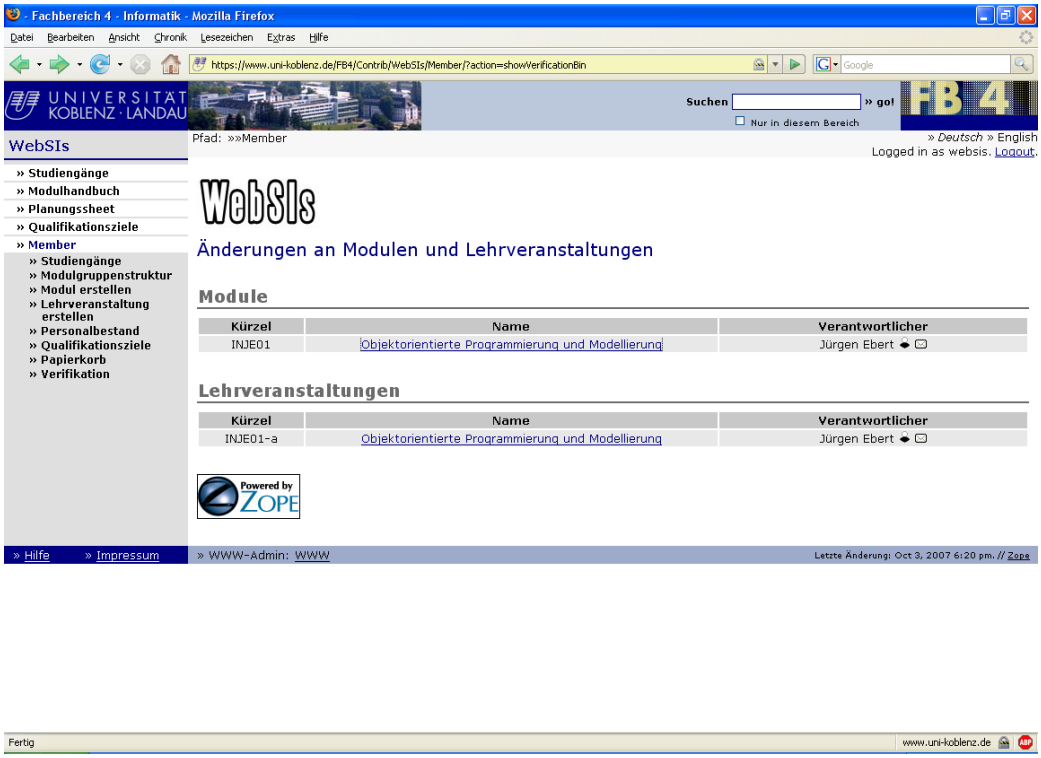

Abbildung 6.17: Verifikation

Der Punkt Verifikation liefert eine Übersicht über alle gemachten Änderungen von Modulen und Lehrveranstaltungen, welche noch nicht vom Dekan verifiziert wurden (siehe Abbildung 6.17). Durch einen Klick auf den Namen der jeweiligen Veranstaltung gelangt man zu dem Formular aus Unterabschnitt 6.1.7 bzw. Unterabschnitt 6.1.8. Der Unterschied zu den normalen Formularen besteht jedoch darin, das nun auch noch die Änderungen zwischen der offiziellen und der geänderten Version angezeigt werden. Abbildung 6.18 verdeutlicht dies nochmals. Am Ende des Formulars befinden sich zwei Buttons zum Ablehnen bzw. Annehmen der gemachten Änderungen.

Rot steht dabei für gelöschte und blau für neu hinzugefügte Wörter.

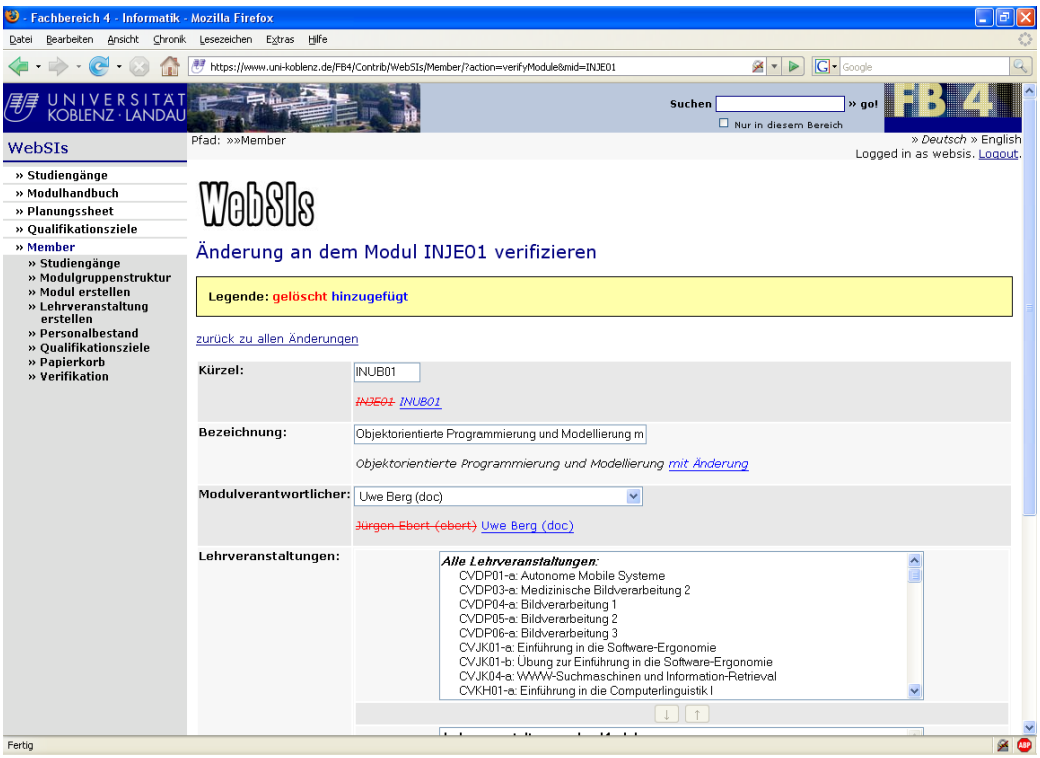

Abbildung 6.18: Verifikation einer Veranstaltung

6.2 Arbeiten direkt auf der Datenbank

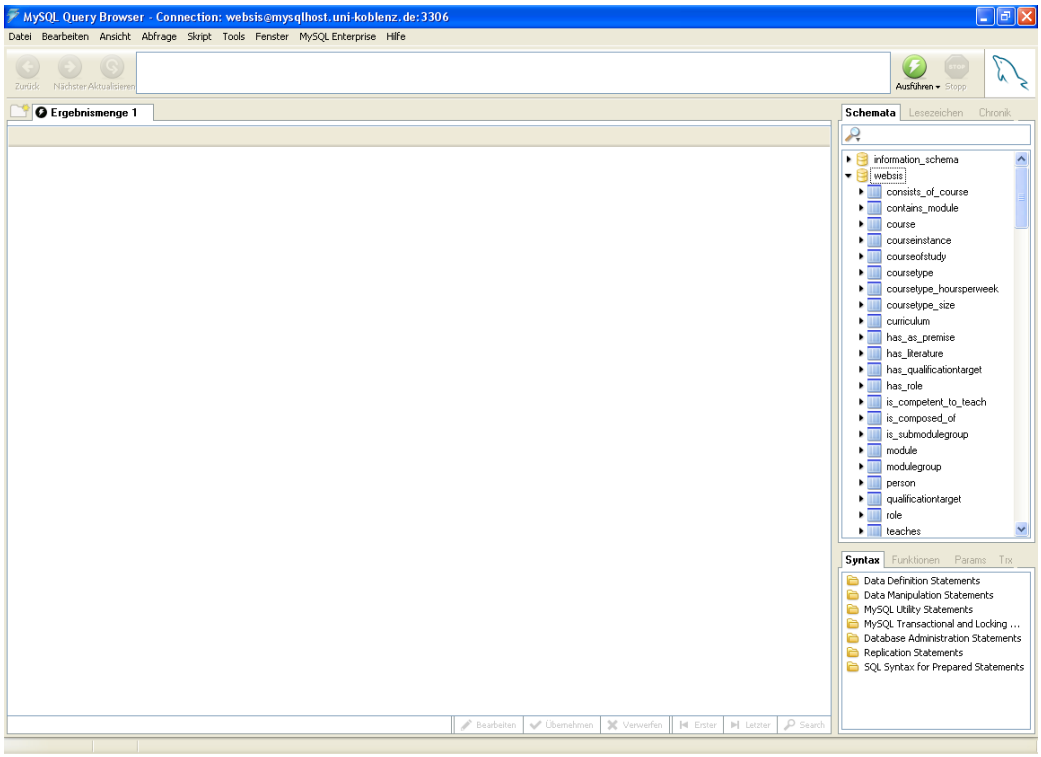

Abbildung 6.19: MySQL Query Browser

Die Benutzer dürfen mit den jeweiligen Rechten auch direkt auf der Datenbank Anfragen stellen bzw. Daten eingeben. Hierfür sollte der *MySQL Query Browser*, welcher Teil der *MySQL GUI Tools*[1] ist, verwendet werden (siehe Abbildung 6.19). Hierfür werden die jeweiligen Datenbankpasswörter benötigt.

6.2.1 SQL-Queries

Generell lässt sich jede denkbare Anfrage mit den jeweiligen Rechten stellen. Bevor jedoch damit begonnen werden kann, eine Anfrage zu stellen, muss noch die richtige Datenbank ausgewählt werden. Dies geschieht durch einen Rechtsklick auf die entsprechende Datenbank im Fenster „Schemata“ im rechten Fensterbereich. Dort wählt man nun den Punkt „zum Vorgabeschema machen“ aus.

Nun kann oben in dem Eingabefeld die SQL-Query eingegeben werden. Mit einem

[1] `http://dev.mysql.com/downloads/gui-tools` (abgerufen am 04.11.2007)

Klick auf den Button „Ausführen“ wird die Anfrage gestellt und unten darunter im Ergebnisfeld angezeigt.

```
1 SELECT cosid, cosname
2 FROM courseofstudy
3 ORDER BY cosname ASC;
```

Auflistung 6.1: Beispiel einer SQL-Query

Die Anfrage in Auflistung 6.1 liefert alle Studiengänge mit Id und Namen in aufsteigender Reihenfolge des Namens.

6.2.2 Gespeicherte Prozeduren

Im Grunde sollte jede sinnvolle Anfrage auch als eine gespeicherte Prozedur existieren. Die gespeicherten Prozeduren finden sich im „Schemata-Fenster“. Welche Variablen die gewünschte Prozedur verwendet, lässt sich mit einem Rechtsklick auf diese und dann einem Klick auf „Prozedur bearbeiten “ erkennen.

Auflistung 6.2 gibt das selbe Ergebnis wie die Anfrage aus Auflistung 6.1 zurück.

```
1 CALL get_all_coursesofstudy(1,0,"");
```

Auflistung 6.2: Beispiel einer gespeicherten Prozedur

7 Fazit und Ausblick

Das im Rahmen der Studienarbeit entwickelte System „WebSIs“ dient der Visualisierung und der Pflege des Modulhandbuchs des Fachbereichs 4 (Informatik) der Universität Koblenz-Landau.

Der öffentliche Bereich der Anwendung bietet allen Studierenden und Studieninteressierten eine übersichtliche Visualisierung des Modulhandbuchs. Dabei können Informationen zu den Modulen und den darin enthaltenen Lehrveranstaltungen einzelner Studiengänge abgerufen werden. Zudem kann das Fachbereichsangebot für gewünschte Semester- /Jahreskombinationen und Studiengänge berechnet werden. Die für die einzelnen Modulgruppen definierten Qualifikationsziele können ebenfalls eingesehen werden.
Der private Bereich der Anwendung, welcher durch eine entsprechende Authentifizierung im System zugänglich ist, dient im Allgemeinen der (zentralen) Pflege des Modulhandbuchs. Modulverantwortliche können dabei die Daten eigener Module und der darin enthaltenen Lehrveranstaltungen editieren. Diese Änderungen müssen dabei zunächst vom Studiendekan verifiziert werden, um „offiziell“ im System eingetragen werden zu können. Weitere Aufgabenbereiche des Studiendekans sind die Editierung von Studiengängen, von Modulgruppen und des Personalbestands. Der Studenplanersteller ist für die Erstellung von Lehrveranstaltungsinstanzen zuständig.

Bei der Entwicklung und speziell der Modellierung des Systems wurde großer Wert auf ein klar durchdachtes, gut strukturiertes und erweiterbares Datenbankmodell gelegt. Die Implementierung wurde dabei objektorientiert mit PHP 5 durchgeführt, wobei auch hier auf eine gute Struktur und eine nachvollziehbare Modularisierung des Quellcodes geachtet wurde.

Eine zentrale Anforderung ist die Integration des Sytems in das uniweite CMS-System (ZOPE), welche letztlich - wenn auch nicht ganz unproblematisch - gelungen ist. Fehlende Features sind die Integration der uniinternen LIT-DB in das System, die Auswertung von Kapatitätsdaten und die automatische Generierung von individualisierten Curriculi, welche aufgrund technischer und zeitlicher Probleme nicht im Rahmen dieser Studienarbeit umgesetzt wurden.

In Zukunft soll das System (durch studentische Hilfskräfte) gepflegt und um fehlen-

de/ weitere Funktionalitäten erweitert werden. Es wird dabei weiterhin als zentrales (Online-)System zur Pflege des Modulhandbuchs des Fachbereichs 4 zum Einsatz kommen.

Literaturverzeichnis

[Bun04] Bundesministerium für Bildung und Forschung: *Bologna-Prozess bringt Hochschulreform voran.* Januar 2004. – http://www.bmbf.de/_media/press/akt_20040129-015.pdf (abgerufen am 09.05.2007)

[Eur06] Europäische Kommision: *Europäisches System zur Übertragung und Akkumulierung von Studienleistungen (ECTS).* Mai 2006. – http://ec.europa.eu/education/programmes/socrates/ects/index_de.html (abgerufen am 09.05.2007)

[He07] He, Wei: *PHParser - A Gateway to PHP!* Webseite, Oktober 2007. – http://www.zope.org/Members/hewei/PHParser/1.1.5/readme (abgerufen am 02.12.2007)

[KE04] Kemper, Alfons ; Eickler, André ; Roth, Margit (Hrsg.): *Datenbanksysteme - Eine Einführung.* Oldenbourg, 2004 (5. Auflage)

[LPMS] Latteier, Amos ; Pelletier, Michel ; McDonough, Chris ; Sabaini, Peter ; Zope Community (Hrsg.): *The Zope Book.* 2.6. Zope Community. – http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/ZopeBook-2_6.pdf (abgerufen am 02.12.2007)

[MyS07] MySQL: *MySQL 5.0-Referenzhandbuch (revision: 7062 vom 2007-07-10).* 07 2007

[Uni06a] Universität Koblenz-Landau: *Bachelorstudiengänge am Fachbereich Informatik an der Universität Koblenz-Landau.* November 2006. – http://www.uni-koblenz.de/FB4/Studying/Bachelor (abgerufen am 09.05.2007)

[Uni06b] Universität Koblenz-Landau: *Masterstudiengänge am Fachbereich Informatik der Universität Koblenz-Landau.* November 2006. – http://www.uni-koblenz.de/FB4/Studying/Master (abgerufen am 09.05.2007)

[Uni07] Universität Koblenz-Landau: *Gemeinsame Prüfungsordnung für Studierende der Bachelor- und Masterstudiengänge des Fachbereichs Informatik an der Universität Koblenz-Landau.* März 2007. – `http://www.uni-koblenz.de/~kgt/POBScMSc.pdf` (abgerufen am 15.06.2007)

[Weg07] Wege ins Studium: *Studienstruktur und -inhalte der Bachelor- und Masterstudiengänge.* Mai 2007. – `http://www.wege-ins-studium.de/studienstruktur.html` (abgerufen am 09.05.2007)