Operative Prozessunterstützung durch Integration einer Prüfungsverwaltung in das LMS-OLAT

Daniel Gerber
daniel.gerber@me.com

Bachelorarbeit

Universität Leipzig
Fakultät für Mathematik und Informatik
Abteilung für betriebliche Informationssysteme
7. September 2009

Betreuer:
Prof. Dr. Hans-Gert Gräbe

Zusammenfassung

Diese Bachelorarbeit befasst sich mit der Entwicklung und Integration einer Prüfungsverwaltungssoftware in das Learning Management System OLAT. Dabei wird, ausgehend von den speziellen Anforderungen am Institut für Informatik der Universität Leipzig, die entstandene Softwarearchitektur und deren Integration in OLAT ausführlich vorgestellt sowie in geringerem Maße auch deren Implementierung veranschaulicht. Zusätzlich wird zur Qualitätssicherung für dieses und weitere OLAT-bezogene Projekte ein Testkonzept und ein Konfigurationsmanagement im softwaretechnischen Sinne entwickelt und vorgestellt. Zur Überprüfung der erbrachten Leistung wird die Software abschließend evaluiert.

Inhaltsverzeichnis

1 Einleitung

Die Prüfungsverwaltungssoftware XMAN, Kurzform für e**X**am **MAN**ager, ist ein an der Universität Leipzig beziehungsweise an deren Institut für Informatik entstandenes Softwareprodukt, das die Durchführung und Planung von Prüfungen vereinfacht. Der Anstoß für diese Arbeit wurde durch das Softwaretechnikpraktikum im Sommersemester 2007 gegeben. Dabei wurden erstmals periodisch wiederkehrende operative Prozesse, wie die Registration von Studenten zu einer Prüfung, benannt, spezifiziert sowie in Software abgebildet. Auf der Grundlage dieser Prozesse ist dann in einem iterativen Prozess die hier vorgestellte Software entstanden, die die beteiligten Akteure entlasten und für einen fließenderen Informationsaustausch sorgen soll. Die aktuelle Version von XMAN wurde erfolgreich im Sommersemester 2009 zur Durchführung von Prüfungen im Bachelorstudiengang genutzt und ersetzt damit das bisher am Institut für Informatik eingesetzte Anmeldesystem, welches im Abschnitt 2.1 näher beschrieben wird. Als technische Grundlage wird das Learning Management System (LMS) OLAT (siehe Abschnitt 2.2) verwendet, welches zugleich auch als Organisationsplattform für eine Vielzahl von Vorlesungen am Institut genutzt wird.

1.1 Zielsetzung

Ziel der vorliegenden Arbeit ist die Erstellung einer Softwarearchitektur, die es Mitarbeitern des Prüfungsamtes und den verantwortlichen Angestellten der einzelnen Abteilungen ermöglicht, einfach und effektiv Prozesse zur Prüfungsdurchführung online zu verwalten. Dabei soll ebenfalls herausgestellt werden, wie sich eine solche Software respektive die vorhandenen Anforderungen möglichst nahtlos in das zu Grunde liegende OLAT System integrieren lassen. Zusätzlich soll die Softwarequalität durch den Entwurf eines an OLAT angelehnten Testkonzepts verbessert und im Hinblick auf zukünftige Entwickler beziehungsweise Entwicklungen ein Konfigurationsmanagement definiert werden. Ferner wird eine Evaluation der hier erbrachten Leistung durchgeführt, um die Akzeptanz der Nutzergruppe gegenüber dem entwickelten Produkt zu ermitteln.

1.2 Gliederung der Arbeit

Diese Arbeit gliedert sich im Wesentlichen in zwei Teilbereiche. Im ersten Hauptbereich, dem Kapitel 4, wird eine Übersicht über die entstandene Architektur der Prüfungsverwaltungssoftware dargelegt, wobei hier insbesondere die in sich abgeschlossenen Module, wie Prüfungen und elektronische Studentenakten sowie deren Integration in OLAT im Vordergrund stehen. Zusätzlich wird an dieser Stelle auch ein Ausblick auf in Zukunft durchführbare funktionale Erweiterungen dieser Architektur gegeben.

Der zweite Hauptbereich befasst sich mit den beiden Softwaretechnikdisziplinen *Validierung und Verifikation* in Kapitel 5 und dem *Konfigurationsmanagement* in Kapitel 6. Dabei stellt die Analyse des bereits als praxistauglich erwiesenen Test- beziehungsweise Konfigurationsmanagements der OLAT-Projektgruppe der Universität Zürich die Grundlage für die Erstellung dieser Dokumente im Kontext einer Nutzung an der Universität Leipzig dar.

Zusätzlich werden diese beiden Bereiche von einer Reihe kleinerer Kapitel umschlossen. Dabei handelt es sich im Kapitel 2 um einen Überblick über die verwendeten Basistechnologien, eine Vorstellung des bisherigen Prüfungsanmeldungssystems und OLAT wie auch der Versuch einer Einordnung der Arbeit in ein softwaretechnisches Vorgehensmodell. In Kapitel 3 werden die gestellten Anforderungen an das System erläutert.

Den Abschluss bildet die Evaluation der Software sowie die im Kapitel 8 zusammengefassten Ergebnisse dieser Arbeit.

Ergänzend soll hier angefügt werden, dass im Folgenden Examen als Synonym für Prüfung verwendet und außerdem auf die Beistellung der weiblichen Form zur männlichen verzichtet wird. Dies geschieht ausschließlich aus Gründen der besseren Lesbarkeit.

2 Grundlagen

2.1 Bisheriges Prüfungsanmeldungssystem

Die Fakultät für Informatik der Universität Leipzig zählt zu den wenigen Fakultäten die eine vollständige Onlineanmeldung für Prüfungen anbietet. Um dies zu gewährleisten wurde vor mehreren Jahren eine einfache Skript-basierte Onlineregistration implementiert und in die Homepage des Prüfungsamtes integriert. Für Studenten begrenzt sich der Funktionsumfang aber lediglich auf die An- beziehungsweise Abmeldung zu Prüfungen. Dabei muss ein Student für jede Prüfung erneut seine persönlichen Daten wie Studiengang, Name, Vorname, Matrikelnummer und so weiter in einer einfachen Eingabemaske angeben. Abgesehen von diesen redundanten Eingaben kann der Nutzer nicht einwandfrei nachweisen, dass er wirklich der ist, der er ausgibt zu sein. So wäre es jedem Nutzer beispielsweise möglich, nach einer Sammlung von persönlichen, aber nicht geheimen Informationen, fremde Studenten zu Prüfungen zu registrieren. Außerdem werden die eingegebenen Informationen nicht auf Übereinstimmung zwischen den einzelnen Informationen wie Matrikelnummer und Name hin überprüft. Auf der Seite der Nutzergruppen Prüfer und Mitarbeiter des Prüfungsamtes wird nach Ablauf der Anmeldefrist eine Datei zum Informationsaustausch aus der Liste der registrierten Studenten generiert und von einem Mitarbeiter des Prüfungsamts zu den zuständigen Professoren per Email manuell weitergeleitet. Hat die betreffende Prüfung nun stattgefunden, wird diese Datei von den Professoren mit den Noten der Kandidaten angereichert und wieder per Email an das Prüfungsamt zurückgesendet. Dieser Austausch wird solange wiederholt, bis keine Probleme mehr aufgetreten sind. Anschließend werden die Noten in die zentrale Datenbank der Universität und die Datenbank des Prüfungsamtes übertragen und stehen dort für spätere Anfragen bereit.

2.2 OLAT

Das Akronym OLAT steht für *Online Learning and Training* und betitelt eine quelloffene und webbasierte Java™Anwendung. OLAT wird seit 1999 an der Universität Zürich entwickelt und eingesetzt und befindet sich momentan in der Version 6.1.1. Es wird in

die Kategorie der Learning Management Systems eingeordnet und ermöglicht webgestütztes Lernen, Lehren und Moderieren durch die Bereitstellung eines umfassenden Kursmanagementsystems. So können beispielsweise neben einem sehr feingranularen Gruppen- und Rechtemanagement diverse Kollaborationswerkzeuge wie Foren, Wikis und Instant Messaging in Kurse eingebunden werden. Auch verschiedene Authentifikationsmethoden, wie Shibboleth oder LDAP und der Einsatz im Clusterverband, sind möglich. Außerdem werden definierte Erweiterungspunkte angeboten und gängige E-Learning Standards wie IMS Content-Packaging[1], IMS-QTI[2] 1.2 und SCORM[3] 1.2 unterstützt sowie auch die im XMAN-Projekt verwendeten Basistechnologien genutzt:

Hibernate ist ein unter der GNU Lesser General Public License veröffentlichtes quelloffenes Persistenzframework für Java, dessen Hauptaufgabe es ist, ein Objekt-relationales Mapping anzubieten. Das heißt, es ist damit möglich, Java-Objekte in relationalen Datenbanken abzubilden und aus diesen Datensätzen bei Bedarf wieder Java-Objekte zu generieren. Außerdem wird durch die dazu verwendete Hibernate Query Language von der darunter liegenden relationalen Datenbank abstrahiert.

Velocity ist ein von der Apache Software Foundation unter der Apache License 2.0 veröffentlichtes quelloffenes Softwareprojekt. Velocity ist eine Java-basierte Template (Schablone) Engine, die eine Template Sprache zur Referenzierung von im Quellcode definierten Objekten zur Verfügung stellt. Somit kann wie im Model-View-Controller Pattern [GHJV95] vorgegeben, eine strikte Trennung von Applikationslogik und Präsentationsschicht gewährleistet werden.

2.3 Eingliederung in Vorgehensmodelle

Da zu Projektbeginn die Liste der Anforderung noch nicht vollständig beziehungsweise manche Anforderungen auch noch unbekannt waren und der Zeithorizont bis zur Fertigstellung eines ersten Produktivsystems kein schwergewichtiges Modell zuließ, war es zu diesem Zeitpunkt nicht möglich, ein spezielles Vorgehensmodell auszuwählen. So kam es zu einer nicht klar definierbaren Vermengung von Merkmalen aus beispielsweise agilen Entwicklungsmodellen wie Extreme Programming, objektorientierten und auch funktionsgetriebenen Modellen.

[1] http://www.imsglobal.org/content/packaging/
[2] http://www.imsglobal.org/question/
[3] http://www.adlnet.gov/Technologies/scorm

3 Anforderungen

Ein essentieller Teil der Erstellung eines Softwareprodukts in einem softwaretechnischen Kontext ist die Erhebung von Anforderungen an dieses System. Eine Anforderung bezeichnet dabei eine Aussage über eine zu erbringende Leistung dieses Produkts. Außerdem werden Anforderungen in funktionale, also die Funktionen, die es bieten soll und nicht funktionale Anforderungen, die Eigenschaften des Produkts, eingeteilt. Diese Anforderungen wurden im XMAN-Projekt anfänglich einmal erhoben und mit dem Fortschreiten der Entwicklungsarbeiten stetig erweitert und überarbeitet. Die folgende Auflistung soll ausgehend auf den in [Grä09] vorgestellten Analyseergebnissen, einen Überblick über die finale Version dieser Anforderungen bieten, die somit die Grundlage für die im nächsten Kapitel vorgestellte Produktarchitektur bildet.

Rollenkonzept

Das von OLAT vorgegebene Rollenmanagement soll um eine neue Rolle *Prüfungsamtmitarbeiter* erweitert werden. Dabei soll der von OLAT vorgegebene Arbeitsablauf zur Änderung der Benutzerrollen so modifiziert werden, dass bereits mit OLAT vertraute Benutzer diesen ohne Kenntnis der Prüfungsverwaltungssoftware benutzen können.

Prüfungsverwaltung

Es soll das vorhandene OLAT System um einen weiteren Reiter *Prüfungsverwaltung* erweitert werden. Dabei soll auf die bereits existierenden Erweiterungspunkte zurückgegriffen werden. An dieser Stelle soll eine zentrale Administration von Studiengängen, Modulen und elektronischen Studentenakten implementiert werden. Zusätzlich muss sichergestellt sein, dass nur Nutzer mit der Rolle *Prüfungsamtmitarbeiter* oder *Administrator* Zugang zu diesem Verwaltungsbereich haben.

Prüfungen

Es soll eine neue Lernressource *Prüfung* angelegt werden. Diese Lernressource soll ebenfalls als Kursbaustein zur Verfügung stehen und eine Unterstützung zur Durchführung für mündliche und schriftliche Prüfungen bieten. Hierbei ist zu beachten,

dass für mündliche Prüfungen mehrere, für schriftliche Prüfungen hingegen lediglich ein Termin angeboten werden darf. Im Allgemeinen werden beide Arten der Prüfung durch eine Menge von Teilnehmern gekennzeichnet, die sich innerhalb eines bestimmten Zeitraums zu einer Prüfung an- beziehungsweise abmelden können. Ferner soll die Möglichkeit bestehen, Prüfungen einem Modul zuzuordnen und eine Option angeboten werden, welche eine vorläufige Prüfungsanmeldung ermöglicht, die durch den verantwortlichen Prüfer oder einen Mitarbeiter des Prüfungsamtes bestätigt respektive abgelehnt werden kann.

Elektronische Studentenakten

Mit der Implementierung der elektronischen Studentenakte soll eine Ergänzung zu den im Prüfungsamt aufbewahrten und in Papierform vorliegenden Akten entwickelt werden. Die Beantragung dieser an genau einen Nutzer gebundenen Akte erfolgt von jedem Studenten selbst, wobei dieser hierbei eine Reihe von persönlichen Informationen wie zum Beispiel Vorname, Nachname und Studiengang wahrheitsgemäß angeben muss. Die Validierung dieser Informationen beziehungsweise der Akte erfolgt vorerst händisch von einem Mitarbeiter des Prüfungsamtes. Außerdem muss die Akte eine Unterstützung zur Verwaltung von Prüfungsergebnissen, Krankenscheinen, Kommentaren vorsehen und die Möglichkeit bieten, Kommunikation zwischen Prüfungsamt und dem entsprechenden Studenten zu dokumentieren und archivieren.

Studiengänge

Um auch den Studiengang in der elektronischen Prüfungsakte zu erfassen, soll die Profilseite von OLAT um eine neues Feld *Studiengang* erweitert werden. Hier ist anzumerken, dass Studiengänge bereits im bisherigen OLAT System als einfache Zeichenkette angegeben werden können, dies aber nicht akzeptabel ist, da so bei einem potenziell sehr großen Benutzerkreis eine Vielzahl von Bezeichnungen für denselben Studiengang entstehen würde. Ferner muss eine Administrationsoberfläche für Mitarbeiter des Prüfungsamts geschaffen werden, um die Einträge des Studiengangsfeldes im Profilformular verwalten zu können.

Module

Wie bereits erwähnt, muss eine Prüfungen einem Modul zugewiesen werden. Hierfür muss eine Benutzeroberfläche geschaffen werden, die es erlaubt im laufenden Betrieb des Portals neue Module anzulegen und diese zu editieren. Außerdem müssen für diese Module insbesondere die Modulnummer, der Name, eine Beschreibung und ein bereits an der Prüfungsplattform angemeldeter Benutzer als Modulverantwortlicher

angegeben werden können.

Mehrsprachigkeit

Das von OLAT vorgegebene Konzept zur Internationalisierung soll in allen Komponenten dieser Erweiterung Anwendung finden. Das heißt, es soll mindestens für die deutsche Sprache eine vollständige Übersetzung dieser Prüfungsverwaltung vorliegen. Insbesondere muss hier darauf geachtet werden, dass die bereits angesprochenen Einträge des Studiengangsfeldes des Profilformulars in die am System unterstützten Sprachen übersetzt werden können.

4 Architektur und Implementierung

Die bisherigen Kapitel haben einen groben Überblick über das bisher zur Verwaltung von Prüfungslebenszyklen verwendete Systeme vermittelt. Außerdem wurden die für einen Einsatz an der Fakultät für Informatik erhobenen Anforderungen an eine solche Software beschrieben. In diesem Kapitel soll nun die Frage gestellt und beantwortet werden, wie die Architektur dieser Applikation aussehen könnte, um eben diese Anforderungen zu erfüllen. Insbesondere muss geklärt werden, an welchen Stellen und wie stark sich XMAN in OLAT integriert. Also ob eher eine selbstständige Version entwickelt wird, welche möglicherweise auch ohne einen Großteil der von OLAT zur Verfügung gestellten Funktionalität auskommt, oder eine von OLAT stark abhängige Version, welche aber die von OLAT bereitgestellten Mittel besser ausschöpfen könnte.
Man könnte sich die Frage stellen, ob es möglich ist, eine Architektur für ein Prüfungsverwaltungssystem, im Weiteren als PVS bezeichnet, zu entwickeln, welches in der Lage ist, sämtliche internen Prozesse einer Fakultät, oder zumindest einer Abteilung, bis ins kleinste Detail abzubilden. Geht man von den allgemeinen, im ersten Moment trivial erscheinenden Prozessen, wie etwa der Erstanmeldung zu einer Prüfung ohne Zulassungsvoraussetzungen aus, erscheint dies durchaus noch denkbar. Aber wie ändert sich die Situation bei einer Vielzahl von unbekannten Faktoren, wie der Menge aller möglichen Studiengänge? Was passiert mit bereits registrierten Studenten, die heiraten (Namensänderungen), einer schwangerschaftsbedingten Studienpause oder einem Auslandssemester? Es gibt hier eine Vielzahl von denkbaren Sonderfällen, aber es gibt mit Gewissheit auch eine große Anzahl von Ausnahmesituationen, die niemand zur Planung eines solchen Systems herangezogen hätte. Es ist folglich nur sinnvoll, bereits vollständig etablierte Prozesse zu digitalisieren, da der durch die Entwicklung entstandene zusätzliche Aufwand nur mit Hilfe eben dieser häufig auftretenden Prozesse kompensiert werden kann. Diese Gründe veranlassten somit das Projektteam, die Architektur der PVS XMAN als ein den Nutzer unterstützendes und nicht vollständig in Software abgebildetes System zu entwickeln.

Das folgende Kapitel wird nun Schritt für Schritt, beginnend bei der Registration der Nutzer, bis hin zur Auswertung der elektronischen Studentenakten beim Prüfungsamt, die zu

Grunde liegende Architektur und in einem geringeren Maße auch deren Implementierung aufzeigen und erläutern. Als Grundlage dieser Arbeiten dient hier die Objektorientierte Analyse, welche in einen statischen und einen dynamischen Bereich unterteilt wird. Zum besseren Verständnis der gewählten Reihenfolge bietet die Abbildung 4.1 einen Ausblick auf die kommenden Abschnitte. Hierbei werden alle nicht weiter kapselbaren, also selbstständig existierenden Module der Reihe nach[1] vorgestellt und in einem abstrakteren beziehungsweise detaillierteren Kontext beleuchtet.

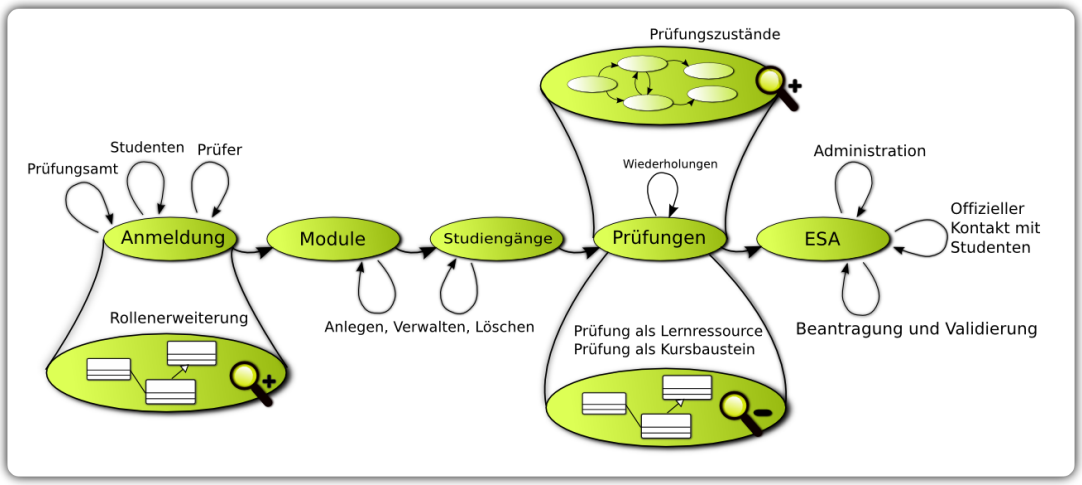


Abbildung 4.1: Überblick der vorgestellten Komponenten und Prozesse

4.1 Gesamtübersicht

Wie in Abbildung 4.2 zu sehen ist, besteht XMAN aus zwei großen Teilsystemen. Das ist zum einen die Komponente *Prüfung*, welche aus einer Menge von Unterkomponenten wie zum Beispiel Modulen, Terminen und dem Examen selbst besteht, und zum anderen die Komponente *Elektronische Studentenakte*, die wiederum aus Unterkomponenten wie beispielsweise eine Menge an persönlichen Informationen, Kommentaren und Krankenscheinen zusammengesetzt ist. Zu diesen persönlichen Informationen zählen der Vor- und Nachname, die Matrikelnummer, der Studiengang sowie die Studserv-Emailadresse[2]

[1] Die Reihenfolge ergibt sich hier aus der bei der Installation vorgegebenen logischen Erzeugung der verwendeten Objekte. So benötigt man um Prüfungen anzulegen Module, die wiederum nur von Mitarbeitern des Prüfungsamtes erstellt werden können und so weiter.

[2] Die Studserv-Emailadresse wird vom Rechenzentrum jedem immatrikulierten Studenten für die gesamte Zeit des Studiums zur Verfügung gestellt.

des jeweiligen Studenten. Die beiden Teilsysteme können theoretisch getrennt voneinander existieren, das heißt es existieren keinerlei Abhängigkeiten zwischen ihnen. Allerdings wäre der Funktionsumfang beider Hauptkomponenten stark eingeschränkt, sollten sie einzeln zum Einsatz kommen. Beispielsweise hätte das Prüfungsamt keinen direkten Überblick mehr über die Noten eines Studenten oder es wäre lediglich möglich, über Studenten Kommentare und Krankenscheine zu erfassen anstatt deren Prüfungsergebnisse zu verwalten. Die Kopplung der beiden Systems an OLAT ist hingegen sehr ähnlich. Die Komponenten *Prüfung* und die *elektronische Studentenakte*, im Folgenden auch ESA oder ESF (engl.: electronical student file) genannt, sind beide sehr stark in die Hauptapplikation integriert und nutzen die zur Verfügung gestellte Funktionalität voll aus. Welche Funktionen damit genau gemeint sind und wie die Integration in OLAT im Detail funktioniert, wird in den folgenden Abschnitten ausführlich beschrieben.

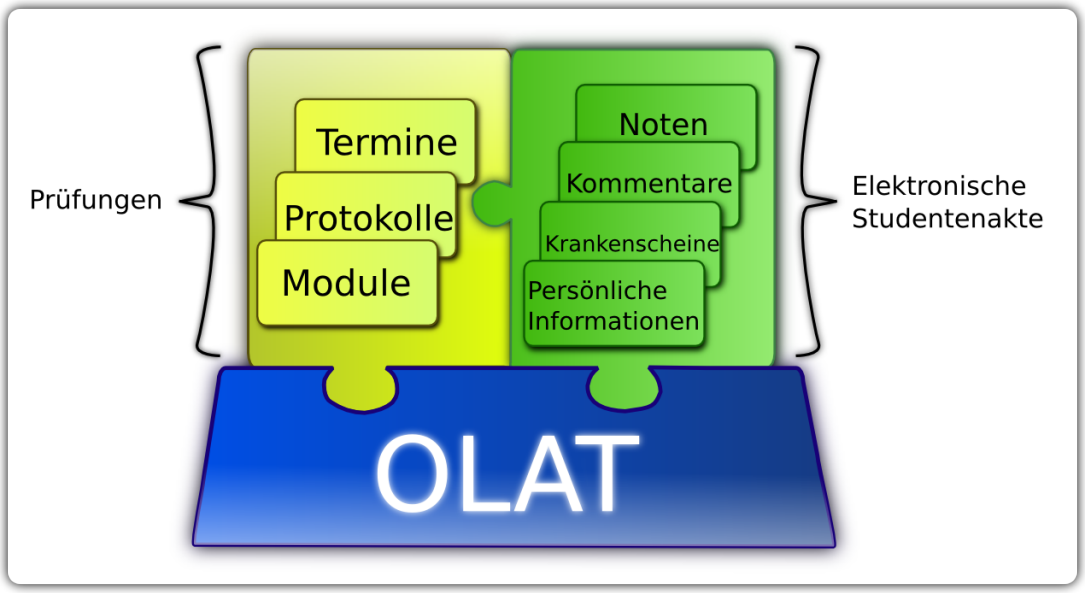


Abbildung 4.2: Veranschaulichung der Hauptkomponenten von XMAN

4.2 Registration und Authentifikation

Registration bezeichnet im Allgemeinen das Eintragen von Informationen in ein Verzeichnis. Im Kontext einer Webapplikation, insbesondere einer Prüfungsverwaltung, versteht man unter Registration die Beschreibung der eigenen Identität, um Informationen abrufen zu können, die nicht registrierten Benutzern verwehrt bleiben. Außerdem kann durch die

Identifizierung der Identitäten eine Zuordnung von real lebenden Menschen mit digitalen Identitäten stattfinden und somit ausgewählten respektive in einer bestimmten Art und Weise dafür qualifizierten Benutzern spezielle Rechte beziehungsweise Rollen zugeordnet werden. Wie aus den Anforderungen in Kapitel 3 hervorgeht, benötigt eine Prüfungsverwaltung mindestens folgende Rollen:

Prüfling:
Eine Person in der Rolle des Prüflings ist ein regulär an der Universität Leipzig immatrikulierter Student, der eine oder mehrere Prüfungen absolvieren möchte.

Prüfungsautor:
Eine Person in der Rolle des Autors, insbesondere hier eines Prüfungsautors, ist ein in einer Abteilung der Universität angestellter Mitarbeiter oder eine externe Hilfskraft, der zu einer in der universitären Lehre angebotenen Veranstaltung eine Prüfung durchführt und für alle dabei notwendigen Prozesse verantwortlich ist.

Prüfungsamtmitarbeiter:
Eine Person in der Rolle eines Mitarbeiters des Prüfungsamtes ist im Kontext dieser Prüfungsverwaltung derjenige Mitarbeiter des Prüfungsamtes einer Fakultät der Universität, der für die Weiterverarbeitung[3] der bei einer Prüfung erzeugten Ergebnisse verantwortlich ist.

Administrator:
Eine Person in der Rolle des Administrators ist ein Mitarbeiter des Rechenzentrums der Universität Leipzig beziehungsweise ein Mitarbeiter der zentralen Dienste der Fakultät oder eine externe, dafür qualifizierte, Hilfskraft, die für die Wartung und Pflege des PVS zuständig ist.

Allerdings ergab sich bereits zu Beginn des Projektes aus der Vertrautmachung mit OLAT, dass diese explizit geforderten Rollen in dieser Form nicht zur Verfügung stehen. Auch ein Mapping von bereits vorhandenen auf die gewünschten Rollen kam nicht in Frage, da dies zu nicht sauber getrennten Zuständigkeitsbereichen geführt hätte. Als Folgerung dieser Tatsachen musste das bestehende Rollenkonzept von OLAT genauer untersucht und anschließend um die fehlenden Bestandteile erweitert werden. Diese Untersuchung ergab ein statisches Rollenkonzept mit einer Menge von fest definierten und unveränderlichen Rollen.

[3] Unter Weiterverarbeitung wird hier die Überführung der Noten in den zentralen Datenspeicher der Universität Leipzig verstanden sowie die Behandlung der mit Nichtbestehen einer Prüfung resultierenden Konsequenzen.

Dazu zählen OLAT-Administrator, Benutzermanager, Gruppenmanager, Autor und der Gast[4]. Es galt nun die fehlende Rolle des *Prüfungsamtes*, genauer eines Mitarbeiters des Prüfungsamtes, für die Erweiterung verfügbar zu machen und ebenfalls dafür zu sorgen, dass einem Benutzer des Systems diese Rolle zur Laufzeit der Prüfungsverwaltungssoftware zugewiesen werden kann. Auf Grund der in diesem Punkt nicht erweiterbaren Architektur von OLAT sind alle folgenden Schritte mit einer Veränderung beziehungsweise einer Erweiterung der OLAT Klassen selbst verbunden.

Der erste Teil dieser Aufgabe, also das Anlegen der neuen Rolle im System, wird durch das Klassendiagramm in Abbildung 4.3 beschrieben. Der Ausgangspunkt dieses Erzeugungsprozesses ist die Klasse *BaseSecurityModule*, die während des Startvorgangs von OLAT instanziiert und initiiert wird. Hier werden die Sicherheitseinstellungen aus den Konfigurationsdateien geladen und der Singleton[5] *PersistingManager* erzeugt und ebenfalls initiiert. Für den Fall, dass die System-Level-Gruppen und die dazugehörigen Policies noch nicht existieren, werden diese nun angelegt. Dies geschieht einmal und ausschließlich während des Startvorgangs und wird des Weiteren nur von einem Thread aufgerufen. In diesem Zusammenhang versteht man unter den System-Level-Gruppen die verschiedenen Mengen der Benutzer mit gleicher Rolle. Somit ergibt sich für die Erweiterung die neue System-Level-Gruppe *ExamAdmins*. Dieser Gruppe wird nun noch eine entsprechende *Policy* zugeordnet. Durch *Policy* wird hier ein Tripel definiert, welches aus Subjekt, Prädikat und Objekt besteht. Das Subjekt steht stellvertretend für die Systemgruppe, das Prädikat für eine Genehmigung, in diesem Fall *HAS_ROLE* und das Objekt für die in *Constants* definierte *OLATResourcable ORESOURCE_EXAMADMIN*. Die neue Rolle ist somit technisch gesehen vorhanden, kann aber bis jetzt niemandem während der Laufzeit oder außerhalb einer MySQL Konsole zugewiesen werden.
Um den zweiten Teil, also die Zuweisung einer Rolle zu einem bestimmten Benutzer, möglichst einfach und in den bereits von OLAT vorgegebenen Arbeitsablauf zu integrieren, mussten wiederum bestehende OLAT Klassen adaptiert werden. So wurde das Formular *SystemRolesAndRightsForm* um ein zusätzliches Checkbox Element erweitert. Wird diese Checkbox aktiviert, löst dies einen Ablauf aus, dessen zu Grunde liegende Klassen und Methoden in Abbildung 4.4 veranschaulicht werden. Es wird nun die ausgewählte Identität ermittelt und vom *PersistingManager* durch die Erzeugung sowie Persistierung einer *SecurityGroupMembershipImpl* der jeweiligen System-Level-Gruppe, in diesem Fall

[4]Der normale Benutzer des Systems wird hier durch das Fehlen aller existierenden Rollen definiert.
[5]Eine Beschreibung zu diesem Design Pattern kann Unter http://java.sun.com/developer/JDCTechTips/2006/tt0113.html gefunden werden.

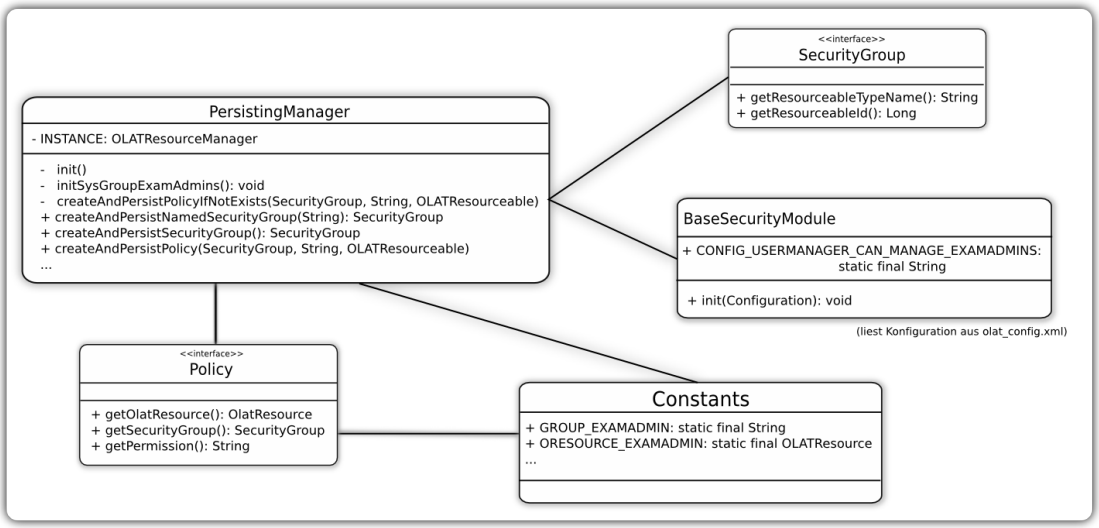


Abbildung 4.3: Klassendiagramm zur Erweiterung der verfügbaren Rollen

der Gruppe *Prüfungsamt*, für unbestimmte Zeit[6] zugeordnet. Von nun an kann die hinter der Identität verborgene Person nach einer erfolgreichen Authentifikation auf die auf das Prüfungsamt beschränkten Methoden und Ressourcen zugreifen. Diese Authentifizierung erfolgt direkt beim Einloggen in das System. Dabei wird für die entsprechende Person respektive für deren Identität geprüft, ob sie sich in den Systemgruppen befindet und ein Objekt *Roles* erzeugt, welches diesen Zustand der Gruppenzugehörigkeit kapselt und für die gesamte Nutzersession zur Verfügung steht.

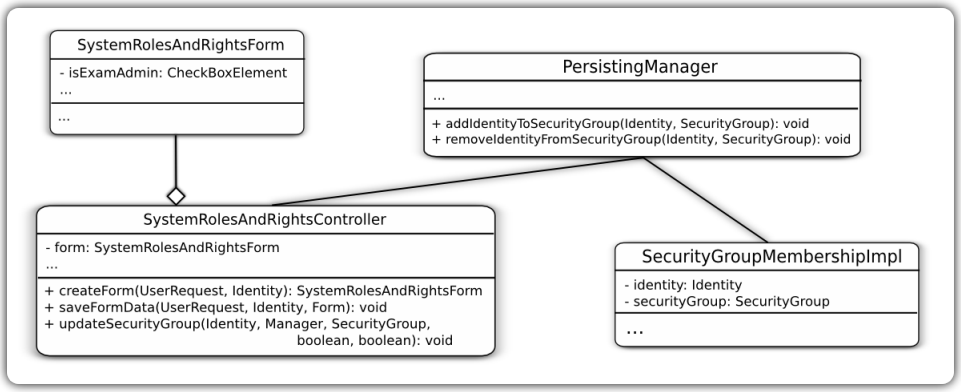


Abbildung 4.4: Zuweisung der Rolle *Prüfungsamt* zu einem Nutzer

[6]Eine Identität verweilt solange in der Rolle *Prüfungsamt*, bis sie ihr von einer höheren Stelle wieder entzogen wird.

4.3 Module

Durch die 1999 von 29 europäischen Bildungsministern unterzeichnete Bologna-Erklärung hat die seit Jahrzehnten größte Studienstrukturreform zur Schaffung eines einheitlichen europäischen Bildungsraums begonnen. Ein Ziel dieses Bologna-Prozesses ist für das Bundesministerium für Bildung und Forschung ([fBF]),

> dass Europa durch die Einführung eines gestuften Studiensystems aus Bachelor und Master mit europaweit vergleichbaren Abschlüssen, die Einführung und Verbesserung der Qualitätssicherung sowie die Steigerung der Mobilität im Hochschulbereich stärker zusammenwächst.

Mit dieser Umstrukturierung geht auch die Modularisierung der Studiengänge und die Bewertung durch Leistungspunkte nach dem European Credit Transfer System einher. Auf Letzteres wird gesondert im Abschnitt 4.7.3 eingegangen. Als Modul wird hier ein zeitlich und inhaltlich abgeschlossener Verbund von Lehrveranstaltungen mit einem ähnlichen inhaltlichen und thematischen Schwerpunkt verstanden. Es ist somit festzuhalten, dass ein Prüfungsverwaltungssystem, welches die aktuellen Standards zur Organisation von Hochschulprüfungen möglichst präzise abzubilden versucht, ohne die Zuordnung der eigentlichen Prüfungen zu Modulen nicht auskommt.
Dies geschieht in der zur jetzigen Zeit aktuellsten Version 1.0.0 direkt beim Erzeugen einer Prüfung. Das dem Examen zugeordnete Modul muss dabei mindestens die Attribute Namen, Beschreibung und einen für alle Prüfungen dieses Moduls verantwortlichen Mitarbeiter des Lehrstuhls besitzen. Optional ist hier zusätzlich noch die Angabe einer dem offiziellen Standard entsprechenden Modulnummer.

4.4 Studiengänge

Ein unverzichtbarer Teil einer Prüfungsverwaltung ist das Erfassen des Studiengangs eines Prüflings. Durch die Vielzahl der momentan noch an der Universität Leipzig beziehungsweise an der Fakultät für Informatik und Mathematik existierenden Studiengänge und die dadurch entstehende Diversität von Prüfungs- respektive Studienordnungen, muss es den Mitarbeitern des Prüfungsamtes möglich sein, die von einem bestimmten Studenten erbrachten Leistungen im Kontext seines angestrebten Studienabschlusses zu bewerten. Somit muss es, wie bereits in Kapitel 3 gefordert, eine Möglichkeit für Studenten geben, ihren Studiengang anzugeben.
OLAT bietet hierfür bereits die Möglichkeit, dies in den persönlichen Einstellungen zu

tun. Allerdings mit der Einschränkung, dass hier nicht aus einer vordefinierten Liste von möglichen Studiengängen ausgewählt werden kann, sondern lediglich ein Freitextfeld[7] zum Eintragen des Studiengangs zur Verfügung steht. Dies ist aber zur eindeutigen Identifizierung der Studiengänge nicht ausreichend. Beispielsweise könnte ein Student, der nach der alten Prüfungsordnung des Bachelorstudienganges studiert, hier bloß *Bachelor* angeben und fälschlicherweise als Student nach neuer Prüfungsordnung aufgefasst werden. Um diese Anforderung dennoch korrekt zu erfüllen, muss also ein alternativer Lösungsansatz gefunden werden. Bei der Recherche zur Lösung dieses Problems wurde schnell klar, dass der eleganteste und flexibelste Weg über den bereits existierenden Erweiterungspunkt *UserPropertyHandler* führt. Hinter diesem Begriff verbirgt sich ein Managerobjekt, welches über genau festgelegte Methoden die Eigenschaften, also auch den Studiengang, eines Nutzers in Formularen darstellen und manipulieren kann, aber auf keinen Fall tatsächliche Nutzerdaten enthält. Eine graphische Übersicht über diese Schnittstelle und deren Implementierung bietet Abbildung 4.5. Außerdem ist anzumerken, dass der *UserPropertyHandler* ein speziell für diesen Zweck bestimmtes Formularelement beispielsweise eine Drop Down Box oder ein Check Box Feld sowie eine Beschreibung des Spaltenkopfes für die Darstellung des Nutzerattributs in Tabellenform liefert. Nach dessen Implementierung und der in Abbildung 4.5 ebenfalls dargestellten Beandefinition kann die neue Nutzereigenschaft in allen Formularen direkt eingebunden werden, ohne sich an diesen Stellen um die Darstellung oder die Manipulation dieser Information zu kümmern.
Um nun auch eine Auswahl an Studiengängen anbieten zu können, wurde im Prüfungsamtbereich eine Möglichkeit zur Erzeugung und Umbenennung jener Daten zur Verfügung gestellt. Diese Editierfunktion beschränkt sich hier allerdings auf die vom jeweiligen Mitarbeiter eingestellte Standardsprache. Da OLAT diverse Funktionen zur Mehrsprachigkeit anbietet, können die Bezeichnungen der Studiengänge direkt über das zur Laufzeit ausführbare Übersetzungsprogramm in alle im System angebotenen Sprachen übersetzt werden.

4.5 Prüfungen

Die bisherigen Abschnitte, insbesondere Module und Studiengänge, haben lediglich einen Rahmen geschaffen, den eine jede Prüfungsverwaltungssoftware zur komfortablen und gut strukturierten Durchführung der eigentlichen Prüfungen dringend benötigt. Außerdem

[7] An der Universität Zürich beziehungsweise dem Authentifikation und Autorisation Infrastruktur Shibboleth Verbund der Schweiz wird der Studiengang über LDAP und Shibboleth im System zur Verfügung gestellt und muss/darf von den entsprechenden Nutzern nicht verändert werden.

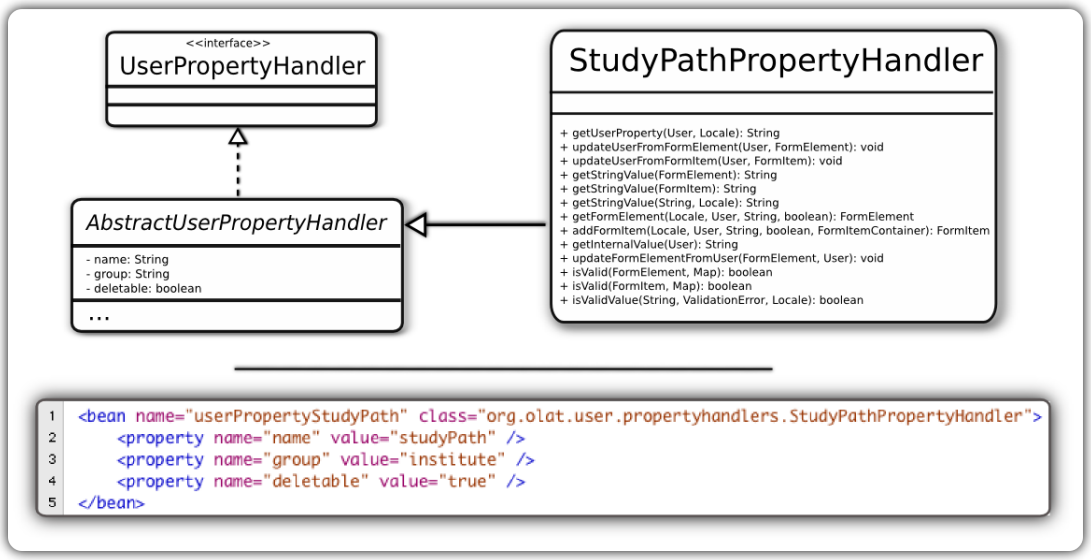


Abbildung 4.5: Implementierung des *AbstractUserPropertyHandler*

wurde bis jetzt noch nicht klar, was genau unter einer Prüfung zu verstehen ist und durch welche Eigenschaften sie sich auszeichnet. So könnte man bis jetzt zu der Schlussfolgerung gelangen, dass es sich bei allen an einer Universität durchgeführten, mit einer Note bewerteten Veranstaltungen, um Prüfungen handelt. Tatsächlich wird aber im Kontext dieser Arbeit eine Prüfung wie folgt definiert:

> Als Prüfung wird eine schriftlich oder mündlich durchgeführte benotete Leistungsmessung verstanden, welche von einem oder mehreren Individuen, in diesem Fall rechtmäßig immatrikulierten Studenten, über einen fest definierten Zeitraum und ein vorbestimmtes Thema sowie zu einem im Voraus veröffentlichten Termin durchgeführt wird.

Aus dieser Definition geht hervor, dass beispielsweise zu referierende Seminare oder durchzuführende Praktika nicht als Prüfung im hier definierten Sinne aufgefasst und deshalb nicht weiter betrachtet werden. Eine weitere offene Frage ist das Konzept der Integration einer solchen Prüfung in das dafür nicht vorgesehene OLAT Wirtssystem. Wie anfänglich erwähnt, existieren wieder zwei Möglichkeiten der Erweiterung. Das wäre einerseits die Integration über ein relativ zu OLAT selbstständiges Subsystem oder andererseits über die Nutzung der bereitgestellten mächtigen Werkzeuge zur Verwaltung von Kursen sowie anderen Lernressourcen. Die Wahl fiel hier, wie auch in den vorangegangen Entscheidungen, zugunsten der tieferen Verbindung zwischen OLAT und XMAN. Dies hat beispielswei-

se auch den Vorteil, dass Personen, die nicht mit XMAN, aber mit dem OLAT System selbst bereits vertraut sind, einen leichteren Zugang zur Prüfungsverwaltung erhalten. In den folgenden Abschnitten soll nun die Umsetzung dieses Konzeptes vorgestellt und die Architektur zur Erfüllung der in Kapitel 3 definierten Anforderungen beschrieben werden.

4.5.1 Prüfung als Lernressource

Die Hauptbestandteile des Learning Management System OLAT sind Lernressourcen. Der Begriff Lernressourcen wird dabei synonym für die Anlage aller verfügbaren Lerninhalte, zu erreichen über den Reiter '*Lernressourcen*', sowie für die an dieser Stelle einzeln abgelegten Lerninhalte verwendet. Ferner werden verschiedene Typen, wie etwa Kurse, Tests oder Wikis unterschieden. Insbesondere ist die Lernressource *Kurs* in der Lage, beliebig viele Kursbausteine[8], wie beispielsweise ein Forum zu beinhalten und damit fähig, Vorlesungen, Seminare oder Praktika abzubilden. Somit würde nach obiger Definition der Prüfung nahe liegen, eine Instanz einer Prüfung als Kursbaustein eines vorhandenen Kurses anzulegen und somit eine direkte Verbindung zwischen Lehre und Leistungsmessung zu schaffen. Allerdings wird dieser Zusammenschluss durch die Grundvoraussetzung des Prüfungsausschusses der Fakultät für Mathematik und Informatik der Universität Leipzig, die für die Lehrveranstaltungsverwaltung zuständige Betriebliche Informationssysteme OLAT Instanz strikt von jeglicher Art von Prüfungsverwaltung zu trennen, unmöglich. Daraus ergibt sich die einzige akzeptable Lösung, für die Prüfungsverwaltung eine eigene Instanz zu erzeugen, die eine Prüfung als Lernressource interpretiert. Dies hat aber auf technischer Seite den Nachteil, dass OLAT für eine solche Form der Adaption keinen Erweiterungspunkt zur Verfügung stellt und wie bereits das Hinzufügen einer neuen System Level Gruppe (Rolle) Veränderungen am OLAT Quellcode erforderlich machen würde. Um die notwendigen Veränderungen möglichst präzise zu erläutern, müssen folgende Begrifflichkeiten im Vorfeld definiert werden.

RepositoryEntry:

Ein *RepositoryEntry* ist eine, wie der Name bereits vermuten lässt, Repräsentation eines Eintrags in der Lerninhaltsablage. Dabei kapselt er nicht etwa die eigentlichen Daten einer Lernressource, sondern vielmehr deren Metainformationen wie den Autor, das Erstellungsdatum oder die Zugriffsrechte.

RepositoryMainController:

Der *RepositoryMainController* ist dafür zuständig, beliebige Lernressourcen zu er-

[8] Das Thema Kursbausteine wird ausführlich im nächsten Abschnitt behandelt.

zeugen und sie ebenfalls wieder zu löschen. Er dient außerdem dazu, bestehende Lernressourcen zu importieren, einen Überblick über existierende Lernressourcen zu geben und eine dem Typ der Lernressource entsprechende Suche bereitzustellen.

RepositoryHandler:

Der *RepositoryHandler* ist ein Managerobjekt, das dafür zuständig ist, die zu einer bestimmten Lernressource gehörenden Controller für das Hinzufügen, Editieren, Starten und Löschen bereitzustellen. Außerdem dient er dazu, zu einer Lernressource gehörende Metainformationen, wie beispielsweise die Möglichkeit zum Download, zu liefern.

Wie auch in Abbildung 4.6 zu sehen ist, musste ein neuer *RepositoryHandler* für die Lernressource beziehungsweise den *RepositoryEntry* Prüfung angelegt werden. Dieser muss, um im gesamten System zur Verfügung zu stehen, in der *RepositoryHandlerFactory* registriert werden. Die eigentliche Implementierung *ExamHandler* der Schnittstelle *RepositoryHandler* stellt nun die für die Erzeugung, das Editieren und Starten der Prüfung notwendigen Controller zur Verfügung. Wird nun beispielsweise im *RepositoryMainController* der Befehl zum Erstellen eines neuen Examens empfangen, leitet dieser den Aufruf an den *RepositoryAddController* weiter, der den entsprechenden *IAddController*, also in diesem Fall den *ExamCreateController* instanziiert. Außerdem existiert ein ähnliches, aber hier nicht weiter beschriebenes Verfahren zur Erzeugung der Instanzen der Controller für den Editier- beziehungsweise Startvorgang.

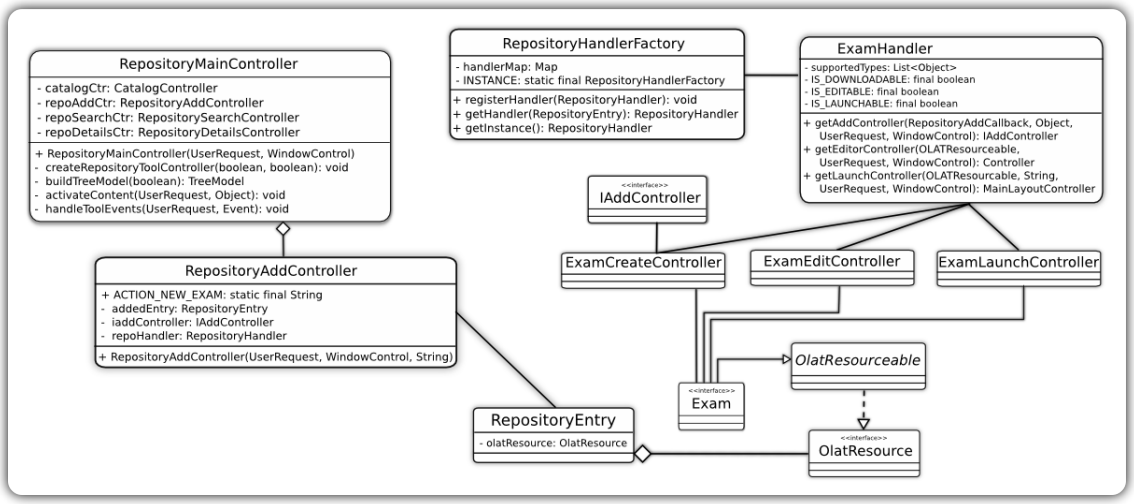


Abbildung 4.6: Prüfung als Lernressource - Notwendige Änderungen

4.5.2 Prüfung als Kursbaustein

Bis jetzt war es nur möglich Prüfungen durch die Veränderungen von OLAT Klassen im Prüfungsverwaltungssystem verfügbar zu machen. Dies könnte aber für manche Benutzergruppen unpraktisch oder ungewollt sein und erfordert somit eine Überarbeitung. Ein alternativer Weg, der nicht dieser schwerwiegenden Einschränkung unterliegt, ist das Einbinden einer Prüfung als Kursbaustein in einen bereits existierenden Kurs. Dies kann zu jedem Zeitpunkt des Lebenszyklus eines Kurses durchgeführt werden, also auch gegen Ende des Semesters bei der Festlegung der Prüfungstermine. Des Weiteren können Kursbausteine beliebig (tief) verschachtelt werden, das heißt, es ist möglich, dass eine Prüfung wiederum eine Prüfung enthält und so weiter. Auch die Zugriffs- und Sichtbarkeitsrechte können für jeden Baustein einzeln und mit Hilfe der so genannten *Experten-Syntax* sehr genau definiert werden. Die Konfiguration der Kursbausteine erfolgt dabei per Konfigurationsdatei und kann zur Laufzeit nicht mehr verändert werden. Wie bereits angedeutet, bietet OLAT für Kursbausteine einen wohldefinierten Erweiterungspunkt, der in Abbildung 4.7 dargestellt wird. So muss zum Erzeugen eines Prüfungskursbausteins eine *ExamCourseNode*[9] implementiert werden. Dieser Knoten wiederum muss sicherstellen, dass zum Editieren und Starten ein *ExamCourseNodeEdit-* beziehungsweise *ExamCourseNodeRunController* zur Verfügung gestellt wird. Die Konfiguration, wie etwa zugehörige CSS Klassen, wird in der *ExamCourseNodeConfiguration* gekapselt.

4.5.3 Struktur

Die Prüfung wurde bis jetzt lediglich in einem globalen Kontext betrachtet. So wurde gezeigt, wie sich eine abstrakt definierte Prüfung in OLAT integrieren lässt sowie welche anderen Komponenten zur Verwaltung von Prüfungen benötigt werden. Nun soll, wie auch im Objektorientierten Entwurf vorgesehen, näher auf die eigentliche Struktur einer solchen Prüfung eingegangen werden. Dieser Abschnitt betrachtet dabei ausschließlich die im Model-View-Controller Design Pattern strikt, von der zur Anzeige und Steuerung benötigten Bestandteile, getrennten Modellklassen beziehungsweise der *Data Access Objects*, die den Zugriff auf die Datenbank kapseln und abstrahieren. Wie auch in Abbildung 4.8 zu sehen ist, haben sich zwei größere Hauptklassen, *ExamImpl* und *ProtocolImpl* sowie einige kleine weitere Hilfsklassen gebildet. Dabei repräsentiert eine Instanz der Klasse *ExamImpl* eine real von einem Prüfungsautor angelegte Prüfung. Die Unterscheidung zwischen schriftlichen und mündlichen Prüfungen geschieht hier über den boolschen Wert

[9] Der Name *Node* entstammt hier von der Baumstruktur des Kurses.

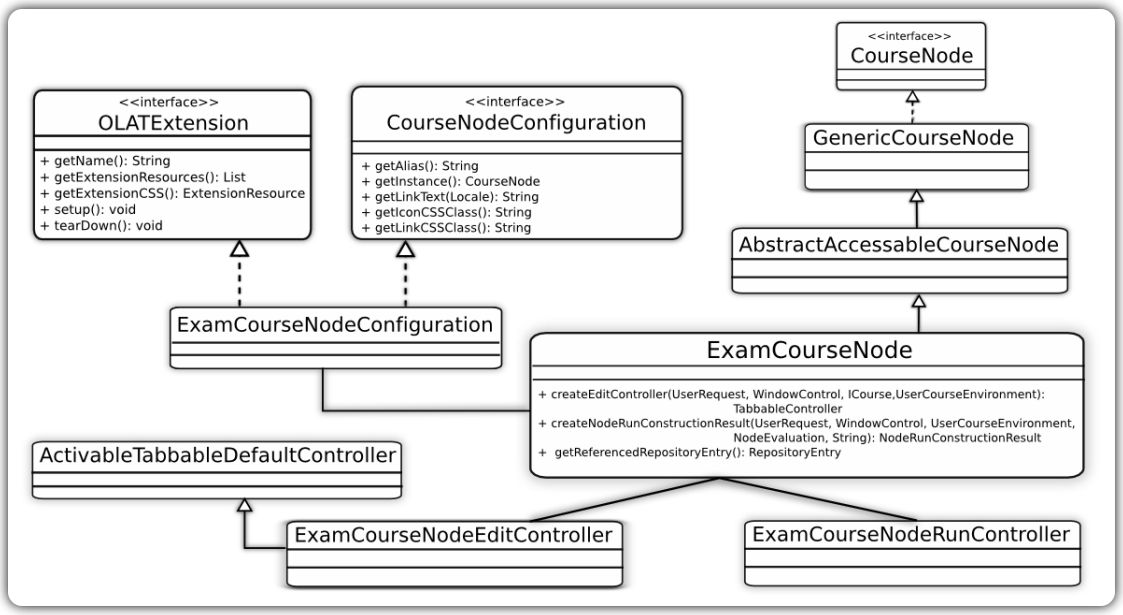


Abbildung 4.7: Prüfung als Kursbaustein - Übersicht

isOral. In einem objektorientieren Modell könnte man davon ausgehen, dass für diese zwei verschiedenen Prüfungsarten auch zwei verschiedene Implementierungen angeboten werden. Tatsächlich wurde dieser Ansatz auch zu Anfang der Entwicklungsarbeiten vorgeschlagen, aber nach einiger Zeit aus Gründen des nicht gerechtfertigten Zusatzaufwands wieder verworfen. Dennoch könnte man, um der Objektorientierung besser zu genügen die Klasse *ExamImpl* als abstrakt definieren und alle, bis auf *getAppointments()* und *setAppointments(List<Appointment> appointments)*, vorhandenen Methoden implementieren. Nun könnten zwei zusätzliche Subklassen *OralExam* und *WrittenExam* nach den gegebenen Anforderungen erzeugt werden, die nur diese beiden Methoden implementieren. Jedem vorhandenen Termin wird genau eine Prüfung zugeordnet, das heißt im Umkehrschluss, dass einer schriftlichen Prüfung ebenfalls genau ein Termin zugeordnet ist (1:1 Beziehung). Einer mündlichen Prüfung hingegen wird eine Menge von Terminen zugewiesen (1:N Beziehung). Es wäre hier ebenfalls möglich gewesen, dass das Examen eine Liste aller sich selbst betreffenden Termine verwaltet, was aber zur Folge gehabt hätte, dass eine zusätzliche Verknüpfungsrelation auf der Datenbankebene für diese Liste der mündlichen Prüfungen angelegt hätte werden müssen. Die zweite große Komponente im Prüfungsmodul bildet die Klasse *ProtocolImpl*. Sie repräsentiert dabei das eher bei mündlichen Prüfungen verwendete, in Papierform vorliegende Prüfungsprotokoll. Die Klasse wird in

dem Moment instanziiert, in dem sich ein Student, hier durch die *Identity* repräsentiert, zu einem bestimmten Termin einer Prüfung anmeldet. In einer früheren Version dieser Software wurde hier das Modul, zu dem die entsprechende Prüfung zugeordnet ist, noch nicht mit erfasst. Das hatte die Konsequenz, dass die an der Abteilung Betriebliche Informationssysteme stattfindenden mündlichen Prüfungen keinem Modul zugeordnet werden konnten. Zur näheren Erläuterung sei hier gesagt, dass es sich bei diesen mündlichen Prüfungen um ein Gemisch handelte, welches zu verschiedenen Prüfungen hätte zugeordnet werden müssen. Mit dieser Änderung ist es nun auch möglich das entsprechende Modul für jeden einzelnen Termin respektive für jedes einzelne Protokoll zu setzen. Außerdem ist es nun nicht mehr notwendig, aber dennoch, möglich einer mündlichen Prüfung ein Modul zuzuweisen. Die vorerst letzte Zustandsänderung des Protokolls erfolgt nach der Absolvierung der Prüfung durch das Eintragen der erreichten Note.

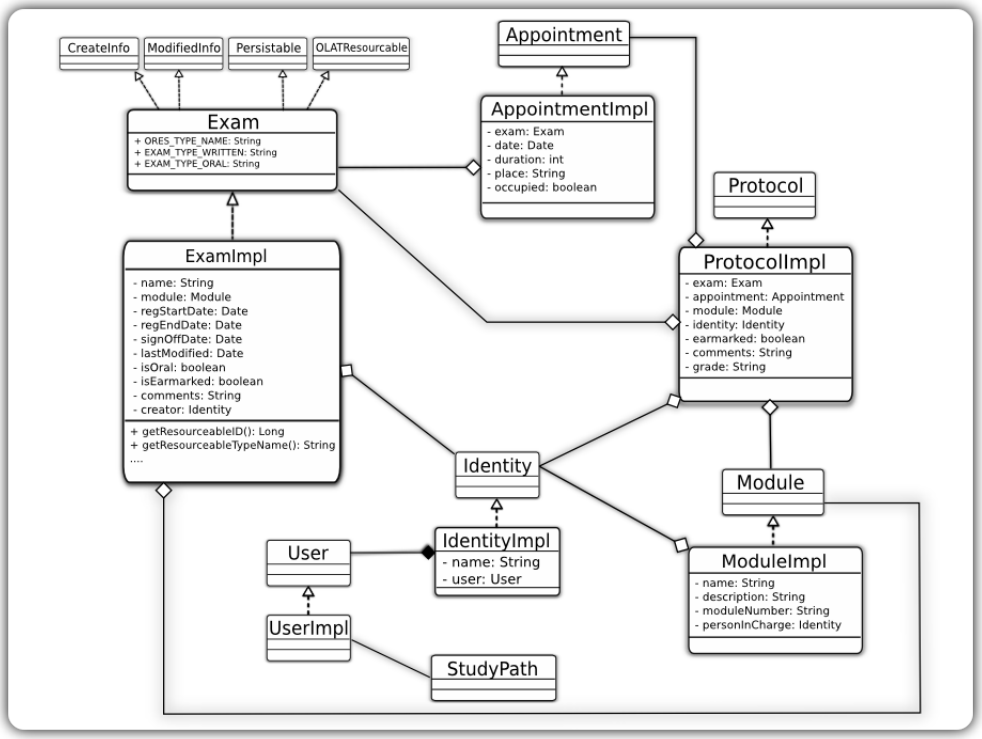


Abbildung 4.8: Prüfung - das Modell

4.5.4 Zustand

Nachdem nun die statische Sicht auf eine Prüfung und die von ihr verwendeten Komponenten betrachtet wurde, wird in diesem Abschnitt die dynamische Sicht, insbesondere der Lebenszyklus einer Prüfungsinstanz, betrachtet. Als Ausgangspunkt dazu dient das in Abbildung 4.9 dargestellte Zustandsdiagramm, welches nun näher beschrieben wird.
Der Lebenszyklus einer Prüfung beginnt mit der Erstellung einer Prüfungsinstanz durch einen dafür autorisierten Benutzer. In der Regel sollten das die in einer Abteilung für den Prüfungsbetrieb zuständigen Angestellten sein. Es wäre allerdings auch denkbar, dass diese Aufgabe, zumindest in der Fakultät für Informatik und Mathematik an der Universität Leipzig, in den Zuständigkeitsbereich der Systemadministratoren wie Herrn Zerbst übergeben wird. Dieser erzeugten Instanz muss zunächst nur die Art der Prüfung, die Modulzugehörigkeit sowie der Name[10] selbst zugeordnet werden. Im Anschluss können nun auch die die Prüfung genauer beschreibenden Details wie die An- und Abmeldefrist, der Prüfungstermin sowie die verwendete Art der Einschreibung, etwa mit oder ohne Vormerken, spezifiziert werden. Ist dies geschehen und wurde der Termin vom Prüfungsamt akzeptiert, kann die Prüfung über den von OLAT vorgegebenen Publikationsprozess veröffentlicht und somit auch für alle Studenten sichtbar gemacht werden. Wurde nun der Beginn der Einschreibung erreicht, können sich die Studenten mit validierter elektronischer Studentenakte einschreiben. Dies kann solange passieren, bis das Einschreibungsende erreicht wurde. Des Weiteren ist es den Prüflingen auch nicht gestattet, sich nach Ablauf des Abmeldedatums von einer Prüfung abzumelden. Dies sowie auch die Anmeldung nach dem Verstreichen der Anmeldefrist kann der Prüfende allerdings manuell veranlassen, falls er damit einverstanden ist. Solche manuell durchgeführten Ein- beziehungsweise Ausschreibungen werden in der elektronischen Studentenakte vermerkt. Zusätzlich kann die Prüfung bis zum Tag des tatsächlichen Stattfindens so editiert werden, dass An- und Abmeldezeiträume sowie alle anderen Attribute beliebig angepasst werden können. Nachdem die Klausur stattgefunden hat und die Korrektur der Prüfung abgeschlossen ist, können die Prüfungsergebnisse den jeweiligen Prüfungsprotokollen zugeordnet werden. Nun folgt die übliche Einsichtnahme der Studenten in ihre Prüfungen. Sollten etwaige Fehler bei der Benotung passiert sein, können die Noten entsprechend verändert werden und ebenfalls Kommentare in dem Prüfungsprotokoll vermerkt werden. Diese werden ebenfalls in der elektronischen Studentenakte eingetragen. Ist die Phase von möglichen Einsprüchen beziehungsweise Notenänderungen beendet, werden die Prüfungsergebnisse an das Prüfungsamt gesendet und die Prüfung ist von nun an beendet und sollte nach einer Frist von einem

[10] Dabei sollten die bereits existierenden Namenskonventionen eingehalten werden.

Jahr gelöscht beziehungsweise extern archiviert werden.

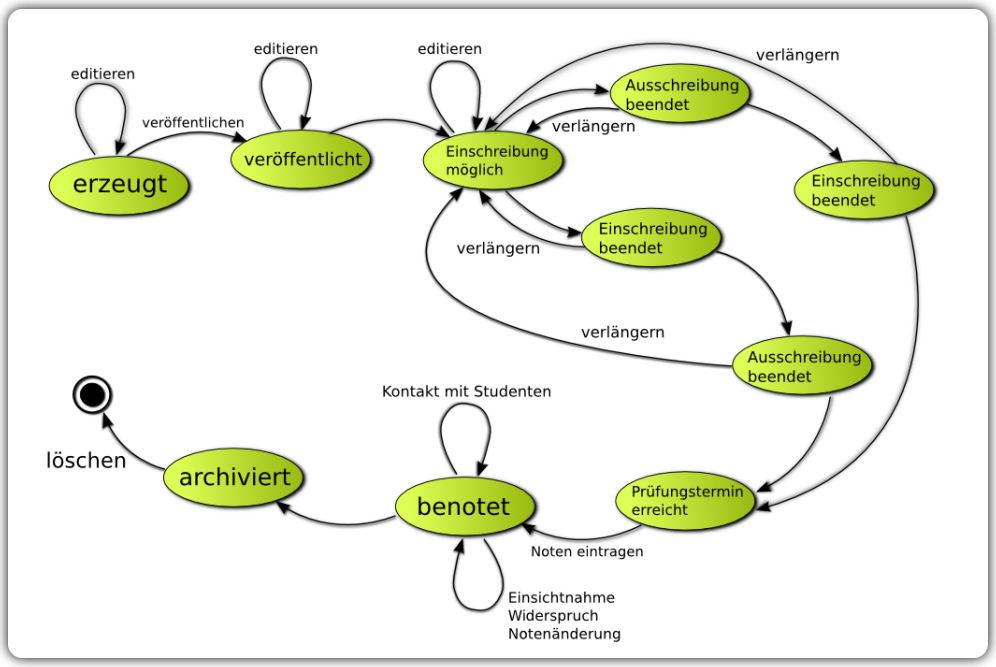


Abbildung 4.9: Zustände einer Prüfung

4.6 Elektronische Studentenakte

Nachdem das Prüfungsverwaltungssystem XMAN im Wintersemester 2008/2009 zum ersten Mal erfolgreich zur administrativen Unterstützung einiger Prüfungen am Lehrstuhl für betriebliche Informationssysteme genutzt wurde, stellte sich heraus, dass zwar einzelne Prüfungen selbst gut durchzuführen waren, aber der zu der Zeit vorhandene Funktionsumfang keinesfalls über diesen speziellen Arbeitsablauf hinaus einsetzbar war. So konnte beispielsweise nicht in einer einfachen und klar definierten Vorgehensweise nachvollzogen werden, welche Prüfungsleistungen ein einzelner Student bereits erbracht hat. Diese Informationen waren zu dieser Zeit über die verschiedenen Prüfungsinstanzen verteilt und an keiner Stelle aggregiert. Ein weiterer, noch gravierenderer Nachteil war, dass praktisch keinerlei Überprüfung der Identitäten durchgeführt wurde. So konnte sich zum Beispiel jede (fremde) Person am System registrieren und sich anschließend in anstehende Prüfungen einschreiben. Um diese Unzulänglichkeiten zu beheben und das Gesamtsystem für eine größere Anwenderschicht attraktiver zu gestalten, wurden iterativ Erweiterungen und

Änderungen an XMAN vorgenommen. Das Ergebnis dieser Iterationen ist die elektronische Studentenakte, im Folgenden auch ESA genannt und die Integration dieser in das bereits bestehende System. In den folgenden Abschnitten wird diese ESA vorgestellt und versucht, einen Überblick über das Zusammenspiel dieser Akte mit den anderen Komponenten des PVS zu vermitteln.

4.6.1 Beantragung und Validierung

Um eine ESA zu beantragen ist es auf der Seite des Studenten nötig, sich am Prüfungsportal zu registrieren. Diese Registration geschieht über den von OLAT vorgegebenen Registrationsprozess, bei dem der am System eindeutige Loginname und ein den OLAT Vorgaben entsprechendes Passwort angegeben werden muss. Möchte ein Prüfling nun an einer Prüfung teilnehmen, das heißt sich zu einem bestimmten Termin einschreiben, so muss er seine persönliche Studentenakte beantragen und validieren lassen. Dazu müssen die Felder *Vorname*, *Nachname*, *Studserv-Emailadresse*, *Matrikelnummer* sowie *Studiengang* wahrheitsgemäß ausgefüllt werden. Anschließend wird vom System eine temporäre, also nicht validierte Studentenakte erzeugt und diese in der Liste der zu validierenden Akten angezeigt. Dieser Vorgang wird in Abbildung 4.10 in der Form eines Sequenzdiagramms dargestellt. Von nun an können dafür autorisierte Benutzer, also Mitarbeiter des Prüfungsamtes beziehungsweise Nutzer mit der Rolle des Prüfungsamtmitarbeiters, die von den Studenten eingegebenen Werte mit den fakultätsinternen oder eventuell auch fakultätsübergreifenden Listen vergleichen und gegebenenfalls Änderungen fordern oder sogar selbst vornehmen. Hierfür kann eine Email an den Studenten geschickt werden, welche in der Akte festgehalten wird. Außerdem haben Nutzer mit der Rolle des Benutzerverwalters zusätzlich die Möglichkeit, Attribute des Studenten über die von OLAT vorgegebenen Änderungsfunktionen zu aktualisieren. Sind alle Informationen des Studenten korrekt, kann die Akte validiert werden, wobei ein entsprechender Eintrag in der Akte abgelegt wird. Von nun an darf sich der Student zu beliebigen Prüfungen einschreiben.
Es ist darauf geachtet worden, dass Studenten unter dem Menüpunkt *Home > Einstellungen* keine persönlichen Attribute verändern können, welche die Korrektheit der Validierung der Studentenakte kompromittieren könnten. Diese Einstellungen werden in der von OLAT dafür vorgesehenen Datei *user_config.xml* vorgenommen. Sollten dennoch Änderungen wie Studiengangwechsel oder heiratsbedingte Änderung des Nachnamens nötig sein, geschieht dies ausschließlich über die Funktion *ESA > Änderung beantragen*. Hierbei werden die Informationen der Akte aktualisiert und diese zur erneuten Validierung dem Prüfungsamt vorgelegt.

Der hier beschriebene Vorgang beruht auf der Authentifizierung durch OLAT, da aber mit der neuesten Version dieser Software auch eine Anbindung an einen externen LDAP Server möglich ist, verändert sich dieser Prozess minimal. Hierauf wird gesondert im Kapitel 4.7.1 eingegangen.

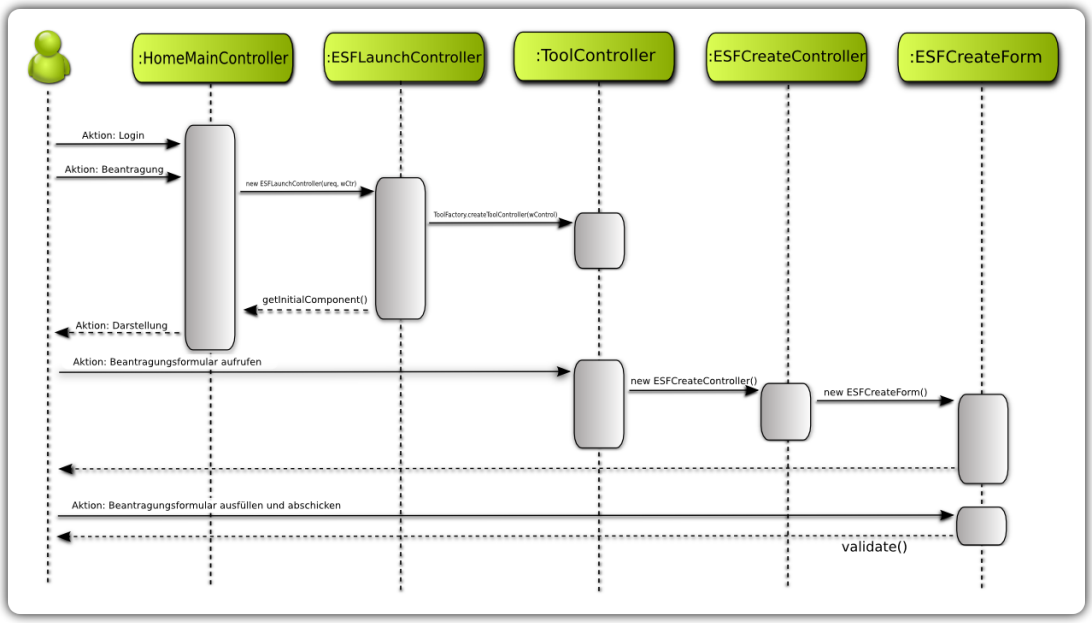


Abbildung 4.10: Sequenzdiagramm - Beantragung einer elektronischen Studentenakte

4.6.2 Aufbau und Darstellung

Um der realen Welt, also den in Papierform vorliegenden Studentenakten des Prüfungsamts, möglichst genau zu entsprechen, wurde die elektronische Version, wie auch in den Anforderungen vorgegeben, in 4 Teile gegliedert. Dieser in Abbildung 4.11 komplett veranschaulichte Aufbau wird nun im Folgenden beschrieben und mit der tatsächlichen Darstellung verglichen.

Persönliche Informationen Diese Art der Information werden, anders als alle restlichen Informationen, nicht von der Prüfungsverwaltungssoftware administriert. Es wird hier lediglich die Referenz auf die im OLAT-System registrierte Identität, also die des Studenten gespeichert. Hierüber ist es möglich, alle persönlichen Informationen über die von

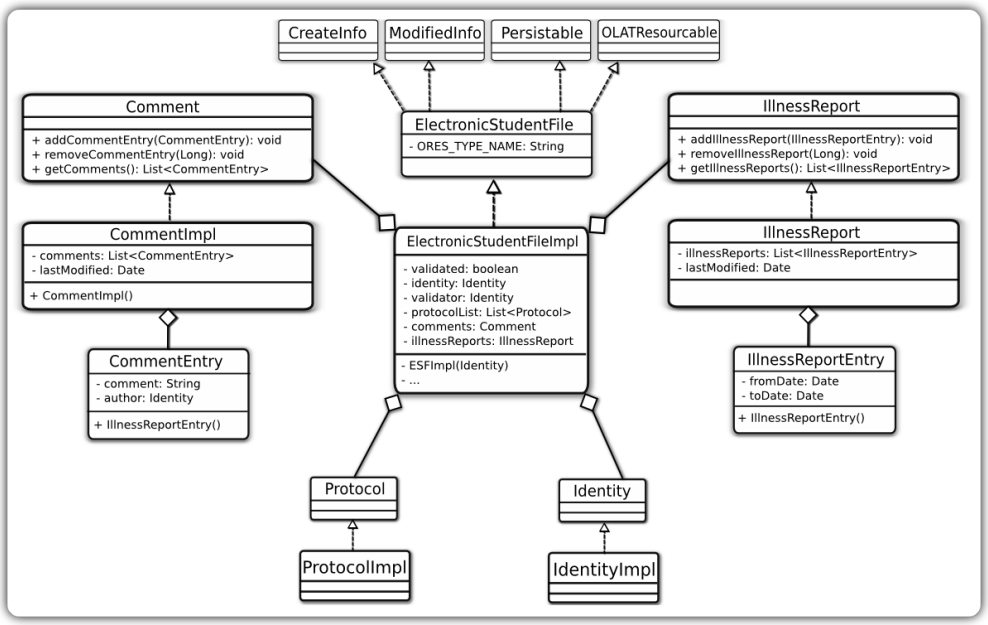


Abbildung 4.11: Klassendiagramm - Die elektronische Studentenakte

OLAT vorgegebenen Möglichkeiten zu speichern beziehungsweise zu aktualisieren. Außerdem wird dadurch vermieden, Daten redundant zu persistieren und somit auch die Konsistenz der Erweiterung gewahrt. Die für diesen Zweck benötigten Attribute werden nun wie folgt dargestellt.

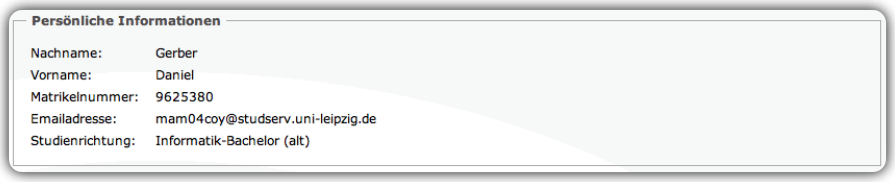
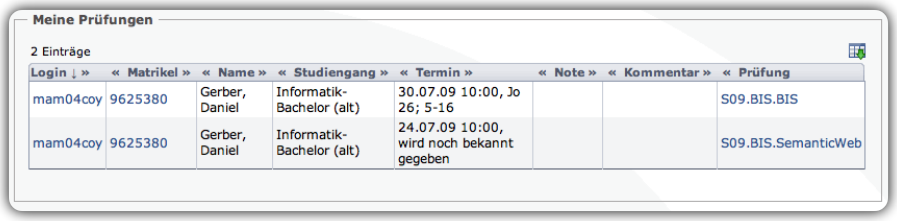


Abbildung 4.12: Persönliche Informationen der Studentenakte

Prüfungsinformationen Wie bereits erwähnt, wurden Prüfungsergebnisse einzelner Studenten in den vorangegangenen Versionen dieser Prüfungsverwaltungssoftware nicht aggregiert. Dieser Nachteil wird nun an dieser Stelle ausgebessert und bietet somit einen Überblick über die bisherigen Prüfungsleistungen des jeweiligen Studenten. Es werden hier alle einer Identität zuordenbaren Prüfungsprotokolle aus der Menge aller Protokolle ausgewählt und angezeigt. Zusätzlich wird ein Link zu den absolvierten Prüfungen und

deren Kommentare sowie Noten, wie in Abbildung 4.13 dargestellt, zur Verfügung gestellt.



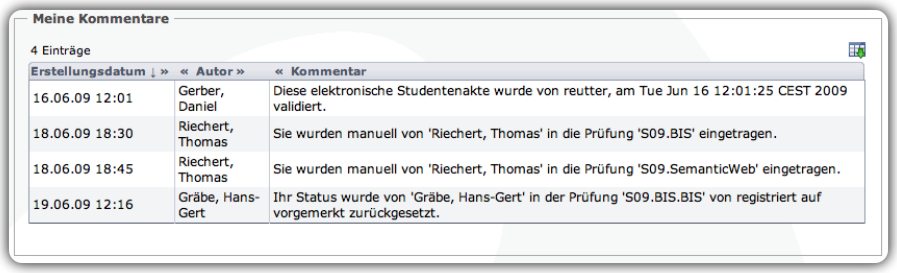
Meine Prüfungen

2 Einträge

Login ↓ »	« Matrikel »	« Name »	« Studiengang »	« Termin »	« Note »	« Kommentar »	« Prüfung
mam04coy	9625380	Gerber, Daniel	Informatik-Bachelor (alt)	30.07.09 10:00, Jo 26; 5-16			S09.BIS.BIS
mam04coy	9625380	Gerber, Daniel	Informatik-Bachelor (alt)	24.07.09 10:00, wird noch bekannt gegeben			S09.BIS.SemanticWeb

Abbildung 4.13: Prüfungsinformationen der Studentenakte

Kommentare Kommentare zu einer ESA unterscheiden sich von den Kommentaren zu einer Prüfung. Hier geht es nicht um präzise Einträge zu einzelnen Klausuren, sondern vielmehr um Informationen, die das gesamte Studium eines Studenten betreffen, beispielsweise zu welchen Prüfungen er sich (nicht) eingeschrieben hat oder ob er den Studiengang gewechselt hat. Zusätzlich kann man hier noch zwischen automatisch generierten und manuell vom Prüfungsamt hinzugefügten Kommentaren unterscheiden, welche sich aber im Äußeren, siehe Abbildung 4.14, nicht unterscheiden.



Meine Kommentare

4 Einträge

Erstellungsdatum ↓ »	« Autor »	« Kommentar
16.06.09 12:01	Gerber, Daniel	Diese elektronische Studentenakte wurde von reutter, am Tue Jun 16 12:01:25 CEST 2009 validiert.
18.06.09 18:30	Riechert, Thomas	Sie wurden manuell von 'Riechert, Thomas' in die Prüfung 'S09.BIS' eingetragen.
18.06.09 18:45	Riechert, Thomas	Sie wurden manuell von 'Riechert, Thomas' in die Prüfung 'S09.SemanticWeb' eingetragen.
19.06.09 12:16	Gräbe, Hans-Gert	Ihr Status wurde von 'Gräbe, Hans-Gert' in der Prüfung 'S09.BIS.BIS' von registriert auf vorgemerkt zurückgesetzt.

Abbildung 4.14: Kommentare der Studentenakte

Krankenscheine Mit Hilfe dieses Bausteins wird die elektronische Studentenakte vervollständigt und Mitarbeitern des Prüfungsamtes die Möglichkeit gegeben, die auf Grund von Krankheit nicht wahrgenommenen Prüfungstermine zu verwalten. Bis jetzt wurde diese Funktion allerdings noch nicht praktisch eingesetzt, was eine Beurteilung der Komponente unmöglich macht. Dennoch können Krankenscheine wie in Abbildung 4.15 angezeigt werden.

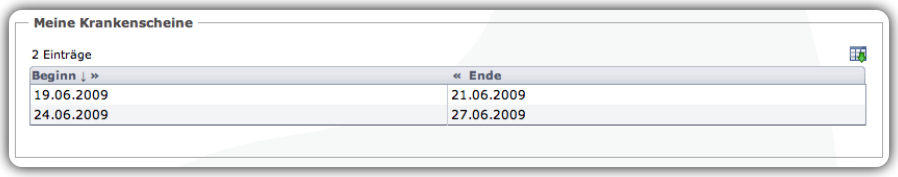


Abbildung 4.15: Krankenscheine in der Studentenakte

4.6.3 Administration der Studentenakten

Das Verwalten von elektronischen Studentenakten ist in XMAN Benutzern mit der Rolle des Prüfungsamtmitarbeiters und des Administrators vorbehalten. Somit kann sichergestellt werden, dass vertrauliche Informationen nur von vertrauenswürdigen Nutzern eingesehen werden können. Die Hauptaufgabe des Prüfungsamtes ist, neben der Verwaltung der bereits erwähnten Module und Studiengänge, die Validierung und die Organisation der Studentenakten. Um dies möglichst intuitiv und schnell durchführen zu können, wurden Tabellen mit *MultiSelectionActions* für die Anzeige der Akten verwendet. Hierdurch können Aktionen wie *Validieren* für eine Vielzahl von Studenten gleichzeitig durchgeführt werden. Außerdem bietet die Funktion *Suchen*, die den *UserSearchController* von OLAT verwendet, eine einfache Möglichkeit, einzelne Akten auch bei einer Vielzahl am System registrierter Studenten zu finden. Andere für die Administration wichtige Funktionen wie das Löschen einer ESA oder die Email mit ESA-Speicherung funktionieren ebenfalls für mehrere Personen gleichzeitig.

4.6.4 Integration und Implementierungsdetails

Der Fokus der bisherigen Kapitel lag zu großen Teilen auf der Architektur der Prüfungsverwaltungssoftware. Wie bereits angedeutet soll in dieser Arbeit der Schwerpunkt auch auf genau dieser Art der Betrachtungsweise liegen. Um dennoch einen groben Überblick über die Verbindung und Abhängigkeiten zwischen OLAT und XMAN zu bieten, wird in den folgenden Abschnitten ausschnittsweise die Verknüpfung der beiden Komponenten und die dazu verwendeten Techniken erläutert. Ferner ist zu bemerken, dass hier nicht die OLAT-Basistechnologien selbst erläutert werden, sondern nur deren Anwendung in der Erweiterung.

Ordnerstruktur Einen Überblick über die in den Quellen herrschende Ordnerstruktur liefert die Abbildung 4.16. Dieses Beispiel für die ESA Komponente soll nicht nur die Ordnung für sich selbst veranschaulichen, sondern vielmehr ein generelles Strukturprinzip für alle

Komponenten des Projektes verdeutlichen. So werden alle nicht Java-Quellcode Dateien, welche direkt im Quellenverzeichnis liegen, in einem mit '_' beginnenden Ordner entsprechend ihrer Funktion verwaltet. Hierbei wird zwischen den Hibernate Mapping Dateien (*.hbm.xml), den Sprachdateien für die Internationalisierung (LocalStrings_de.properties) und den Velocity Templates für die Anzeige unterschieden. Des Weiteren existieren, falls vorhanden, Ordner für Tabellen[11] (table), für Formulare[12] (form), Exceptions und die Controller. Datenobjekte, wie in diesem Fall die elektronische Studentenakte, werden zusammen mit ihrem Interface und dem entsprechenden Manager im Wurzelverzeichnis des Pakets verwaltet.

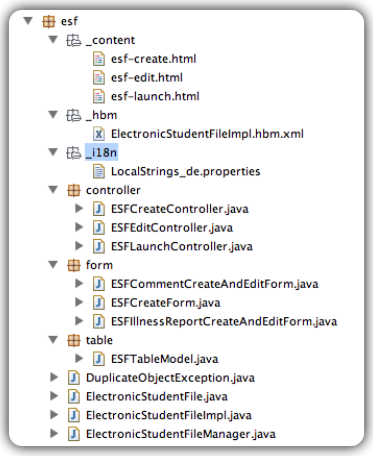


Abbildung 4.16: Ordnerstruktur der ESA Komponente

Manager Zu jedem Datenobjekt, wie in diesem Fall die elektronische Studentenakte, existiert ein spezieller Manager, der auf diesen Objekten arbeitet. Diese Managerklassen folgen somit dem in [SM02] beschriebenen Prinzip, die Daten von den eigentlichen Datenzugriffsmechanismen zu separieren, wodurch die Datenobjekte leicht ausgetauscht werden könnten. Konkreter sind diese Manager für das Erzeugen und Löschen dieser Objekte zuständig, ohne dabei aber ihren internen Zustand zu kennen oder zu verändern. Außerdem implementieren diese Klassen auch die Schnittstelle zur relationalen Datenbank über Hibernate. So können sie etwa Objekte in der Datenbank aktualisieren oder komplexere Anfragen, wie zum Beispiel die Anzahl aller nicht validierten Studentenakten mit einem anderen Studiengang als der Standardstudiengang, an dieselbe stellen. Zusätzlich sind alle

[11] Als Tabellen werden hier Klassen bezeichnet, die von DefaultTableDataModel erben.
[12] Als Formulare werden hier Klassen bezeichnet, die von Form erben.

Manager Singletons, das heißt sie besitzen einen privaten Konstruktor und eine öffentliche sowie statische *getInstance()*-Methode. Somit kann wie in [GHJV95] erläutert, gewährleistet werden, dass nur ein Managerobjekt zur gleichen Zeit mit einem Datenobjekt auf der Datenbank arbeitet.

Controller Wie durch das MVC Prinzip vorgegeben, existieren in den Paketen *admin*, *catalog*, *esf* und *exam* Ordner, die Controller enthalten. Um eine möglichst nah an OLAT angelegte Implementierung zu bieten, existieren für die beiden Datenobjekte Prüfung und elektronische Studentenakte jeweils Controller zum Erzeugen, Ändern und Starten dieser Ressourcen. Im Paket *admin* und *catalog*, dient ein Controller nicht wie eben beschrieben zum Organisieren von Datenobjekten, sondern der Durchführung komplexer Arbeitsabläufe. So dient beispielsweise der Controller *ExamAdminESFController* zur Validierung von Studentenakten, zum Versenden von Emails, dem Suchen von Studentenakten und so weiter, also allen administrativen Aufgaben des Prüfungsamts. Ein einzelner Controller zeichnet sich im Besonderen durch seinen Konstruktor, in dem er beispielsweise dessen *ToolController* initialisiert, ein Velocity-Template lädt und eventuell benötigte Tabellen beziehungsweise Formulare erzeugt, und den beiden *event*-Methoden aus. *Event*-Methoden dienen im Generellen dazu, Ereignisse von anderen überwachten Objekten abzufangen und entsprechend zu behandeln. Der Unterschied zwischen den beiden Hauptmethoden ist hier lediglich die Art der überwachten Objekt. Auf der einen Seite handelt es sich selbst wieder um Controller und auf der anderen Seite etwa um Tabellen oder Formulare. Das folgende Codebeispiel soll verdeutlichen wie diese *event*-Methoden grundsätzlich arbeiten:

```
1  // ein überwachter Controller hat ein Ereignis ausgelöst
2  public void event(UserRequest ureq, Controller ctr, Event event) {
3
4      // der das Ereignis auslösende Controller war der eigene ToolController
5      if ( ctr == this.toolCtr ) {
6
7          // das auslösende Ereignis ist: jemand möchte eine Studentenakte anlegen
8          if ( event.getCommand().equals("action.add.esf") ) {
9
10             ...
11         }
```

Abbildung 4.17: Funktionsweise der *Event*-Methoden

Anzeige OLAT verwendet für die Realisierung der MVC Architektur und der daraus resultierenden Trennung von Applikationslogik und Layout das unter [SJ06] beschriebene

GUI[13] Framework. Um dieses Prinzip auch in den Erweiterungen aufrecht zu erhalten, werden alle GUI-Arbeitsabläufe, wie auch im vorherigen Abschnitt beschrieben, ebenfalls in wiederverwendbaren Controllern gekapselt, wodurch das Layout nur noch durch CSS[14]-Modifikationen und HTML-Fragmente beeinflusst werden kann, wie auch Abbildung 4.18 zeigt.

```
1 <!-- Tabelle mit Prüfungsprotokollen -->
2 <fieldset class="o_whiteBg"><legend><b>$r.translate("esf-edit_html.myExams")</b></legend>
3
4     $r.render("protocolTable")
5 </fieldset>
```

Abbildung 4.18: Auszug eines Velocity Templates

Es ist somit möglich, graphische Inhalte, wie im Fall dieser Prüfungsverwaltungssoftware Tabellen und Formulare, in Controllern zu generieren beziehungsweise deren Zustand zu berechnen und anschließend der *View*-Komponente zu übergeben, welche diese dann entsprechend des zugehörigen *Renderers* in eine HTML-Repräsentation überführt.

4.7 Ausblick

Durch die bereits vorgestellten Komponenten wird den Prüfenden und Mitarbeitern des Prüfungsamts ein flexibles und umfangreiches Werkzeug zur Seite gestellt um operative Prozesse zur Prüfungsverwaltung zu unterstützen. Allerdings kann diese Hilfestellung noch durch eine Vielzahl von Erweiterungen dieser Prüfungsverwaltungssoftware verbessert werden. Hierzu werden im Folgenden drei mögliche Themen vorgestellt, welche eventuell auch im Rahmen des Softwaretechnikpraktikums umgesetzt werden könnten.

4.7.1 Authentifizierung über LDAP

Mit OLAT 6.1.0 ist neben der schon in früheren Versionen vorhandenen Authentifikation der Nutzer durch OLAT selbst und dem Single-Sign-On Service Shibboleth die Möglichkeit zur Authentifikation über einen LDAP-Server hinzugekommen. Der Vorteil dieses Mechanismus ist, dass Nutzer nicht mehr selbst ihre persönlichen Daten eingeben müssen, sondern diese Informationen vom LDAP-Server empfangen werden können. Somit könnte die Beantragung und gleichzeitige Validierung der elektronischen Studentenakte automatisch

[13] Graphische Benutzeroberfläche
[14] http://www.w3.org/TR/CSS21/

ohne Interaktion des Nutzers mit XMAN geschehen, da davon ausgegangen werden kann, dass es nur einen Nutzer geben kann, der die korrekte Kombination aus Nutzernamen[15] und zugehörigem Passwort kennt, nämlich er selbst. Da aber die Datenschutzbestimmungen an der Universität Leipzig lediglich zulassen, die Zugehörigkeit eines Passwortes zu einem Nutzernamen mit ja oder nein zu beantworten und nicht die Abfrage, der für die Erzeugung der elektronischen Studentenakte notwendigen Informationen wie Nachname, Vorname und Studiengang zulässt, muss die Beantragung und Validierung vorerst weiter manuell, wie in Abschnitt 4.6.1 beschrieben, stattfinden. Diese Option wurde für Prüfungen des Sommersemester 2009 erstmalig getestet und muss nach Ablauf dieser Prüfungsperiode evaluiert werden.

4.7.2 Integration der OLATE-Erweiterung

An der Universität Leipzig wird im Moment die in [Fro09] beschriebene OLAT Erweiterung OLATE entwickelt. Dabei handelt es sich um eine Integration von dem ebenfalls an der Universität Leipzig entwickelten Aufgaben-Framework [BW06] zur elektronischen Durchführung von Prüfungen in OLAT. Hier wäre es denkbar, eine OLATE Prüfung als Kursbaustein in eine XMAN Prüfung zu integrieren, oder beide Komponenten zusammen in einen übergeordnetem Kurs zu kombinieren. Dabei könnten Studenten, welche sich zu einer XMAN Prüfung angemeldet haben, direkt zur Elate-Prüfung weitergeleitet werden und die Ergebnisse ihrer absolvierten Prüfungsleistung direkt nach Beendigung der Onlineklausur[16] an XMAN übergeben werden. Hier können nun diese Informationen den Studentenakten der Prüflinge zugeordnet werden und eine Note beziehungsweise entsprechende Kommentare in der Akte vermerkt werden.

4.7.3 Generierung von Transcripts of Records

Wie schon im Abschnitt 4.3 erwähnt wurde, erfolgt die Abrechnung der Studienleistungen in den neuen Bachelor- und Masterstudiengängen durch das European Credit Transfer and Accumulation System (ECTS). Wie auch aus dem Interview mit Herrn Reutter vom Prüfungsamt hervorging, wird zur Fertigung der sogenannten *Transcript of Records*, also den erbrachten Leistungen eines Studenten im Laufe seines Studiums, ein leistungsfähiges und vor allem flexibles Werkzeug zur Erstellung dieser Nachweise benötigt. So könnte man sich beispielsweise vorstellen, dass aus den erzielten und zusätzlich manuell hinzu-

[15] Der Nutzername ist an der Universität Leipzig das Studserv-Kürzel.

[16] Falls die Onlineklausur zuerst von Korrektoren korrigiert werden muss, kann die Ergebnisübergabe erst nach Beendigung dieser Arbeiten stattfinden.

gefügten Prüfungsleistungen über Open Source Java Bibliotheken wie *iText*[17] dynamisch PDF-Dokumente entsprechend den, an der Universität Leipzig benutzten, Vorlagen zu generieren.

[17] http://www.lowagie.com/iText/

5 Testkonzept

Moderne Softwaresysteme wie OLAT sind mittlerweile so komplex und umfassend, dass sie es dem zumeist mehrköpfigen Entwicklerteam äußerst schwer machen, für eine Fehlerfreiheit ihres Produkts zu garantieren. Um den in der Software enthaltenen Anteil an Fehlern dennoch zu minimieren, haben sich in der Softwaretechnik verschiedene Testverfahren herausgebildet und etabliert. In der Definition der **ANSI/IEEE Std. 610.12-1990** ([ans90]) versteht man unter Test

> the process of operating a system or component under specified conditions, observing or recording the results and making an evaluation of some aspects of the system or component.

Das Beobachten und Aufzeichnen sowie die Auswertung einiger Aspekte des Systems kann sowohl durch statische als auch dynamische Verfahren durchgeführt werden. Im Folgenden wird diese Unterscheidung in Bezug auf OLAT und die für OLAT entwickelten beziehungsweise noch zu entwerfenden Erweiterungen genauer untersucht und auf Grundlage dieser Untersuchung ein Testkonzept für die Implementierungsarbeiten an der Universität Leipzig vorgestellt.

5.1 Testkonzept in OLAT

Die von OLAT verwendete Prüfmethode für einen automatisch

> überprüfbaren und jederzeit wiederholbaren Nachweis der Korrektheit eines Softwarebausteines relativ zu vorher festgelegten Anforderungen ([Den])

übernimmt ein Verfahren der testgetriebenen Entwicklung, welches als Unit-Testen bezeichnet wird. Das JUnit Framework, ein Bestandteil agiler Softwareentwicklung, erlaubt es die Klassen beziehungsweise Units der Modellebene auf Fehler zu testen. OLAT beschränkt sich in Sachen Tests ausschließlich auf die Managerklassen, welche die grundlegende Funktionalität der Workflows bereitstellen, zum Beispiel die Persistierung in der

Datenbank, sowie die Objekte, die von den Managern ver- beziehungsweise bearbeitet werden. JUnit-Tests auf View- und Modellebene sind nicht vorgesehen. Das folgende Klassendiagramm veranschaulicht die Einbindung des JUnit-Frameworks in OLAT und gibt einen kurzen Einblick in die Verwendung des Testwerkzeugs.

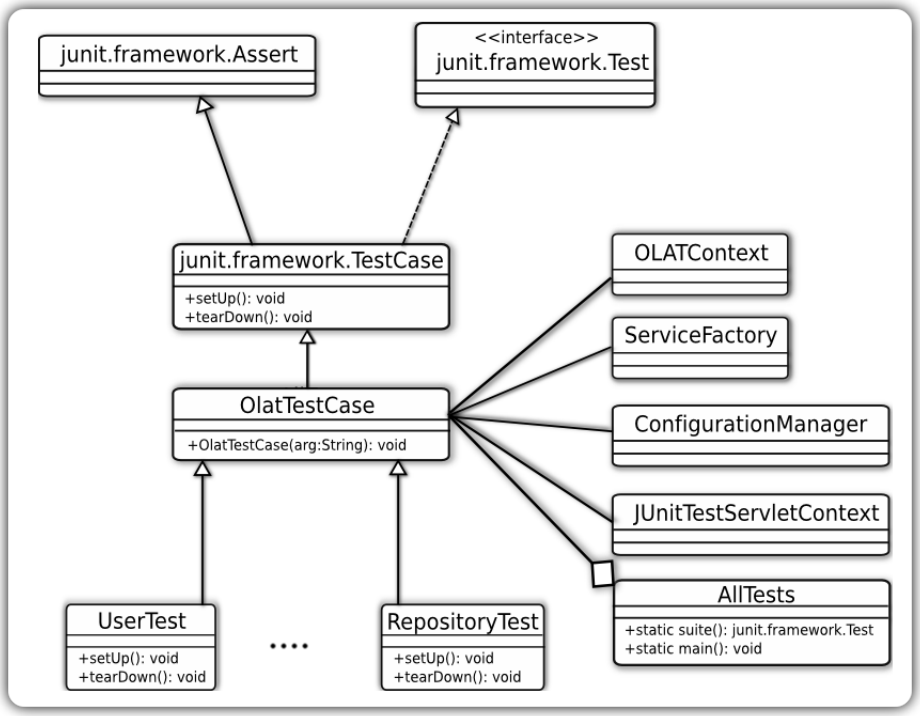


Abbildung 5.1: Klassendiagramm OLAT - Testkonzept

Die Klasse org.olat.core.test.OlatTestCase, die von junit.framework.TestCase erbt, ist Oberklasse eines jeden JUnit Use Case Tests. In einem statisch initialisierten Block werden hier die OLAT-Module über den org.olat.configuration.ConfigurationManager geladen, sowie das Brasato-Framework, also der **Olatcore** initialisiert. Ferner wird dem org.olat.core.-configuration.OLATContext mitgeteilt, das es sich um einen JUnit-Test handelt und ein speziell für JUnit-Tests vorgesehener (Dummy)-org.olat.core.test.JUnitTest-ServletContext erzeugt. Jeder JUnit-Test implementiert dann die Methode **public junit.framework.-test.Test.suite()**, die eine neue junit.framework.TestSuite anhand der JUnit-Test-Klasse erzeugt und alle mit 'test' beginnenden Methoden dieser Klasse in diese Suite aufnimmt. Außerdem werden die Methoden junit.framework.TestCase.setUp() beziehungsweise junit.-framework.TestCase.tearDown() überschrieben. Die setUp-Methode wird vor und die tear-Down-Methode nach dem Ausführen des Test aufgerufen. Üblicherweise werden in setUp()

Klassenvariablen initialisiert und die Testdatenbank bereinigt sowie in tearDown() die Datenbankverbindung geschlossen. Die Containerklasse org.olat.test.AllTests implementiert ebenfalls die **public junit.framework.test.Test suite()**-Methode, in der alle Suiten sämtlicher JUnit-Tests in die lokale Variable **suite** hinzugefügt und zurückgegeben werden. Die eigentliche Auslösung der Testreihen erfolgt über den Befehl 'ant junit' des Build-Tools Ant und erzeugt eine Auswertung der Testfälle im HTML-Format.

5.2 Testkonzept für gegenwärtige und zukünftige Projekte

Nachdem nun das Testkonzept von OLAT untersucht und verstanden wurde, kann eine Strategie zur Entwicklung von beliebigen anderen Projekten mit OLAT-Erweiterungscharakter konzipiert werden. Die folgenden Abschnitte beschreiben, wie sich ein solches Konzept in das bereits vorhanden OLAT Testkonzept einordnen sollte und welche Komponenten es mindestens enthalten muss. Außerdem wird der Aufbau dieser Komponenten an einigen Beispielimplementierungen demonstriert.

5.2.1 Integration in das Testkonzept von OLAT

Da OLAT bereits ein sehr komfortables und einfach zu bedienendes Unit-Test-System bereitstellt, werden die in den Erweiterungen implementierten Testfälle in dieses Konzept eingebunden. Allerdings ist es dazu nötig, einige OLAT-Konfigurationsdateien zu modifizieren. In der build.properties-Datei wird eine neue Eigenschaft 'xman.source.test.dir' hinzugefügt, die den absoluten Pfad zu den für die Erweiterung geschriebenen Testklassen als Wert enthält. In der build.xml-Datei werden zwei neue Ant-Targets angefügt, die für das Erzeugen der erweiterungsspezifischen Testdatenbank und das eigentliche Auslösen aller über einen regulären Ausdruck gefundenen JUnit-Tests verantwortlich sind. Es ist folglich möglich, die graphische Veranschaulichung dieser Testfälle ohne Programmieraufwand zu nutzen.

5.2.2 Standardisierung der Testfälle

Um eine Vereinfachung des Informationsaustausches zwischen Entwicklern und entwicklungsbegleitenden Mitarbeitern sowie eine methodische Vereinheitlichung zu erreichen, werden die Testfälle nach dem folgenden Schema entwickelt.

1. Methodennamen der Testfälle beginnen mit **test**
2. Methodennamen enden mit einem den Testfall treffend bezeichnenden Namen

3. Kommentierung der Testfälle geschieht in englischer Sprache
4. Kommentar beinhaltet:
   - Was wird getestet?
   - Handelt es sich um einen Komponententest oder Integrationstest?
   - Bei Integrationstests werden die integrierten Komponenten genannt.
   - Gibt es Besonderheiten?
5. **assertXXX()**-Anweisungen werden mit der Begründung des zu erwartenden Ergebnisses versehen.

Ein so standardisierter Testfall sollte wie folgt aussehen:

```
1  /* this testcase checks if an appointment is added properly to an exam
2   *
3   * integration test between appointment and exam component
4   *
5   * there are no special characteristics
6   */
7  public void testAddAppointmentToProtocol() throws Exception {
8
9          ...
10
11         /* should be true, if the appointment
12          * is set properly
13          */
14         assertTrue(protocol.getAppointment() != null);
15 }
```

5.2.3 Konfiguration des entwicklungsbegleitenden Testsystems

Bisherige Erfahrungen haben gezeigt, dass zu wenig Feedback der Anwender an das Entwicklerteam weitergegeben wurde. Bisher war den zukünftigen Benutzern noch nicht oder nur in einem geringen Maße ein detaillierter Blick auf den Fortschritt und die Qualität der zu entwickelnden Systems möglich. Dies soll nun durch die Einführung einer Demonstrationsinstanz, welche die komplette Entwicklungsphase begleitet, behoben werden. Hierzu wird eine OLAT-Instanz inklusive der zu erweiternden Funktionalität konfiguriert und der Öffentlichkeit zugänglich gemacht. Die Aktualität dieser Instanz wird durch ein nächtliches Update gewährleistet, welches durch ein Script initiiert wird. Es ist somit möglich, entwicklungsbegleitend Systemtests durchzuführen, eventuell auftretende Fehler von Komponenten und Komponentensystemen schon frühzeitig zu erkennen und dem Entwicklerteam

genaue Änderungswünsche und gefundenes Fehlverhalten mitzuteilen. Nähere Erläuterungen folgen im Kapitel Konfigurationsmanagement.

5.2.4 Komponententests auf Basis einzelner Objekte

Als Objekte werden in diesem Kontext Klassen angesehen, die einen bestimmten Zustand des Gesamtsystems kapseln und ebenfalls in der Lage sind, diesen Zustand mit Hilfe der Manager-Klassen zu persistieren, wieder herzustellen und zu modifizieren. Es ist unerlässlich, dass für das einwandfreie Funktionieren dieser Zustandsänderung alle Attribute solcher Objekte auf mögliches Fehlverhalten getestet werden. Hierfür wird ein TestCase angelegt der beispielsweise wie folgt aussehen kann:

```
1 Appointment app = AppointmentManager.getInstance().createAppointment();
2 Date date = new Date();
3 app.setStartDate(date);
4 assertEquals(date.getTime(), app.getStartDate().getTime());
```

Es ist hier allerdings noch nicht von Bedeutung, Rücksicht auf die durch die Geschäftsprozesse festgelegten Wertebereiche dieser Objektattribute zu nehmen, da diese durch die von OLAT vorgegebene org.olat.core.gui.components.form.Form.validate()-Methode zur Laufzeit kontrolliert und vom Nutzer gegebenenfalls korrigiert werden muss.

5.2.5 Komponententests auf Basis der Eingabeformulare

Ein weiterer wichtiger Testbestandteil ist das Überprüfen der Nutzereingaben. Hier muss sichergestellt werden, dass keine Daten ins System gelangen, welche dort zu einem inkonsistenten Zustand führen können. Um dies zu gewährleisten, wird für jede eigens entwickelte org.olat.core.gui.components.form.Form, also für jedes Eingabeformular, ein TestCase angelegt. Dieser TestCase muss sicherstellen, dass alle Anweisungen der validate()-Methode des Formulars überdeckt und nur eine den Wertebereichen entsprechende Eingabe möglich ist. Eine Ausnahmeregelung gilt lediglich, wenn die validate()-Methode durch ein triviales **return true;** implementiert wird. In diesem Fall ist es nicht nötig einen TestCase zu erzeugen.

5.2.6 Komponententests auf Basis der Geschäftsprozesse

JUnit-Tests werden, wie von OLAT vorgegeben, nur für den Business-Layer der zu entwickelnden Erweiterung geschrieben. Da durch das Brasato-Framework eine strikte Trennung

zwischen Model, View und Controller vorgegeben ist und die Anwendung somit aus wiederverwendbaren Controllern besteht, die die Geschäftsprozesse kapseln, wird klar, dass die Komponententests geschäftsprozess-spezifisch entwickelt und implementiert werden müssen. Das heißt, dass alle in der **public void event(UserRequest ureq, Component source, Event event)** und der **public void event(UserRequest ureq, Controller source, Event event)**-Methoden der Controller abgebildeten Geschäftsprozesse auf enthaltene Fehler hin geprüft werden. Somit wird eine Simulation der Benutzerinteraktion vollzogen. Zur Veranschaulichung der Unit-Tests auf der Ebene der Geschäftsprozesse soll folgendes Beispiel dienen:

– Als zu testenden Geschäftsprozess wird das Erzeugen einer neuen Prüfung definiert. Dieser Vorgang geschieht über den de.xman.exam.controller.ExamCreateController. In dessen event()-Methode wird ein Examen nach den Vorgaben: schriftlich oder mündlich, der zugeordneten Kategorie und dem zuständigen Betreuer der Prüfung angelegt. Dieser Vorgang wird nun in der ***test*CreateWrittenAndOralExam()**-Methode auf möglicherweise enthaltene Fehler überprüft. So sollte:

```
assertFalse(writtenExam.isOral(), oralExam.isOral());
```

zu einem erwarteten Fehlschlag führen.

So wie hier vorgegeben wird mit allen definierten Geschäftsprozessen verfahren. Des Weiteren werden alle in den Manager-Klassen implementierten Methoden auf Fehler überprüft. Sollten hier oder in den Businessworkflowmethoden Fehler gefunden werden, die nicht augenblicklich behoben werden können, werden diese im **ISSUE-Tracking** des Projektes (näher im Teil Konfigurationsmanagement beschrieben) nach den dort herrschenden Vorgaben eingefügt. Solche Issues werden zu den Reviews vom zuständigen Programmierer vorgetragen und gemeinsam werden Lösungen gesucht beziehungsweise Probleme behoben.

5.2.7 Integrationskonzept für Komponenten

Haben die einzeln entwickelten Komponenten einen Unit-Test erfolgreich überstanden, das heißt, sie sind für sich isoliert fehlerfrei funktionsfähig, wird das Zusammenspiel mehrerer Komponenten untereinander getestet. Hierzu wird ein spezielles Testszenario ausgewählt, das im Stande ist sicher zu stellen, ob beide Probanden nach der Zusammenführung selbst noch fehlerfrei funktionieren und das dadurch entstandene Subsystem ordnungsgemäß arbeitet. Die Integration der Komponenten geschieht somit bottom-up, das heißt, die so entstandenen Subsysteme werden im weiteren Verlauf des Testprozesses als erfolgreich

getestete Komponente angesehen. Der Integrationstestprozess endet somit erst, wenn alle Komponenten beziehungsweise Subsysteme durch die bottom-up-Integration zu dem endgültigen Produktivsystem vereint wurden. Zur besseren Veranschaulichung soll folgendes Beispiel dienen:
Ausgangspunkt ist hier, dass die Komponenten Examen und Kategorie erfolgreich ihre Unit-Tests absolviert haben. Es wird nun ein Testszenario ausgewählt, das das Zusammenspiel der Komponenten simulieren soll:

```
1  public void testRetrieveExamsByCategory() {
2      ...
3
4      exam1.setCategory(cat1);
5      exam2.setCategory(cat1);
6      exam3.setCategory(cat1);
7
8      ...
9
10     /* should be true, because there three exams
11        which belong to the first category */
12     List<Exam> examList = ExamDBManager.getInstance().
13                     findExamsByCategory(cat1);
14
15     assertEquals(3, examList.size());
16
17     /* should be true, because there are no exams
18        which belong to the second category */
19     List<Exam> emptyList = ExamDBManager.getInstance().
20                     findExamsByCategory(cat2);
21
22     assertTrue(examList.size() == 0);
23 }
24
```

5.2.8 Durchführung von regelmäßigen Durchsprachen

Aufgrund der mittlerweile großen Vielfalt der an der Universität Leipzig entwickelten Erweiterungen für OLAT werden zukünftig einmal im Monat beziehungsweise bei dringendem Bedarf nach Zuruf Durchsprachen abgehalten. Diese Treffen sollen einen informellen Charakter haben und einen Erfahrungsaustausch zwischen den vielen kleinen Entwicklerteams gewährleisten. Es sollen Probleme, zukünftige Arbeiten und mögliche Interferenzen zwischen den einzelnen Erweiterungen erkannt, besprochen und gelöst werden. Eine besondere Vorbereitung für eine solche Art von Treffen ist im Allgemeinen nicht von Nöten.

Sollte es sich allerdings bei der zu besprechenden Problematik um Fehler aus dem Issue-Tracking des Projektes handeln, ist dafür Sorge zu tragen, dass die Ursache des Problems bekannt, in einer den nicht direkt am Projekt beteiligten Personen verständlichen Form vorzutragen ist und bereits im Vorfeld mögliche Lösungsvorschläge erarbeitet werden. Sollte die Durchsprache zu keiner akzeptablen Lösung führen, wird umgehend ein neuer Termin vereinbart und das Problem auf Grundlage der neu gewonnenen Ergebnisse erneut besprochen. Ergänzend ist festzuhalten, dass einer der Akteure der Durchsprache für die Erstellung und anschließende Veröffentlichung des Durchspracheprotokolls verantwortlich ist.

5.2.9 System- und Abnahmetest der OLAT-Erweiterung

Die System- und Abnahmetests werden durch mehrere Benutzer durchgeführt, die sich an der vollständig integrierten Produktivinstanz einloggen und alle im Vorfeld definierten Geschäftsprozesse ohne Zuhilfenahme des Quellcodes, so genannte black-box-Tests, überprüfen. Da es sich bei den Entwicklungen an der Universität Leipzig in den meisten Fällen um kleine Entwicklerteams handelt, ist es notwendig, dass auch Programmierer selbst an diesen Tests teilnehmen. Als weitere Tester werden hier die Projektbetreuer und eventuell zu diesem Zweck ausgewählte Mitarbeiter des Lehrstuhls genannt. Um einer möglichst breiten Benutzergruppe ein fehlerfreies System zur Verfügung zu stellen, werden die System- beziehungsweise Abnahmetests unter allen gängigen Browsern durchgeführt. Zu diesen Browsern zählen der Internet Explorer, Mozilla Firefox und Safari in der jeweils aktuellsten Version. Die Testdaten für den Abnahmetest werden nach den am Lehrstuhl und der Fakultät herrschenden Richtlinien erzeugt und anschließend getestet. Dieser Vorgang darf die Gesamtdauer von zwei Wochen nicht überschreiten und endet mit der erfolgreichen Abnahme des Projekts oder mit der Beanstandung eines oder mehrerer Geschäftsprozesse mit anschließender Reimplementierung der fehlerhaften Funktionalität.

6 Konfigurationsmanagement

Ein wichtiger Unterstützungsprozess der Softwaretechnik ist das Konfigurationsmanagement, wobei hier die Entwicklung und Anwendung von Verfahren und Standards zur Strukturierung sowie Organisation eines sich fortlaufend weiterentwickelnden Softwareprodukts im Mittelpunkt stehen. Die folgende Definition wurde vom American National Standards Institute[1] in Zusammenarbeit mit der Electronic Industries Alliance[2] erarbeitet:

> Configuration Management ... is a management process for establishing and maintaining consistency of a product's performance, its functional and physical attributes, with its requirements, design and operational information, throughout its life.

Konfigurationsmanagement bietet geeignete Methoden, Werkzeuge und Hilfsmittel an, um die Überwachung des Produkts und dessen Teile innerhalb des Lebenszyklus zu gewährleisten. Des Weiteren soll durch wohldefinierte Änderungen die Weiterentwicklung und Pflege eines Softwareprodukts vereinfacht, dokumentiert und transparent gehalten werden.
Im folgenden Kapitel wird untersucht, wie das Konfigurationsmanagement von OLAT selbst, sowie in kommerziellen Unternehmen, hier am Beispiel der Frentix GmbH[3], definiert ist. Außerdem wird an Hand der hierbei gewonnenen Informationen, ein für die Entwicklungsarbeiten des BIS-Lehrstuhls angemessenes Konfigurationsmanagementkonzept erarbeitet.

6.1 Konfigurationsmanagement des OLAT Projektes

Da OLAT nun mehr seit 1999 von einem 16 köpfigen Entwicklerteam ständig erweitert wird und mittlerweile auf rund 450000 Zeilen Quellcode verschiedenster Programmiersprachen angewachsen ist, ist ein ausgeklügeltes Konfigurationsmanagement zwingend erforderlich. Grundsätzlich werden alle Konfigurationseinheiten im zentralen Concurrent Versions System (CVS) gehalten. Zu diesen Einheiten zählen unter anderem die Dokumentation sowohl

[1] http://www.ansi.org/
[2] http://www.eia.org/
[3] http://www.frentix.com/de/index.html

für Endanwender als auch Entwickler, der OLAT-CORE sowie der OLAT-Quellcode selbst. Die Grundlage für Fehlerverwaltung, Problembehebung und Projektmanagement bietet die von Atlassian entwickelte Instanz der webbasierten Applikation Jira[4]. Jira unterstützt vor allem die Priorisierung, Zuweisung und das Überwachen von Fehlern beziehungsweise Erweiterungen und bietet zahlreiche Statistiken zur Prozessverbesserung. Die ebenfalls von Atlassian entwickelte Applikation FishEye[5] gibt den Entwicklern einen komfortablen Zugriff zu den CVS-Repositories der einzelnen Projekte über eine Webschnittstelle. Außerdem werden alle Entwickler via RSS über Änderungen im Repository auf dem Laufenden gehalten.
Zum eindeutigen Identifizieren von Konfigurationseinheiten, wie zum Beispiel Versionen oder Branches wurden an der Universität Zürich folgende Tagging Konventionen eingeführt:

OLAT-[version]-RC[nummer]-[YYYYMMDD]
: bezeichnet einen Release-Kandidaten (Release-Candidate).

OLAT-[version]-CR[nummer]-[YYYYMMDD]
: bezeichnet eine Quellcode Erneuerung für den Testserver (Code-Refresh).

Die Abfolge der Releases wird durch die folgenden Tags beschrieben:

OLAT-[version]-GM[nummer]-[YYYYMMDD]
: bezeichnet das Golden Master als Grundlage für eine Go/No Go-Entscheidung.

OLAT-[version]-FINAL[nummer]-[YYYYMMDD]
: wird an der Universität Zürich installiert.

OLAT-[version]-PUBLIC[nummer]-[YYYYMMDD]
: wird auf der Homepage veröffentlicht.

Der Entwicklungszeitraum einer Version beträgt in etwa sechs Monate, wobei der Veröffentlichungstermin in der Regel einen Monat vor dem offiziellen Semesterbeginn der Universität Zürich liegt. Einen Monat vor diesem Termin erfolgt ein Code-Freeze und eine ausführliche Testrunde wird eingeleitet. OLAT liegt momentan in der Version 6.1.0 vor. Der Releasetermin für die Beta-Version 6.2.0 ist voraussichtlich der 18.08.2009.
Neue größere Features, welche zum Beispiel in Zusammenarbeit mit der Frentix GmbH oder der Bildungsportal Sachsen GmbH entstehen, werden für ein anschließendes internes beziehungsweise externes Review auf einem Branch entwickelt und nach erfolgreichem

[4] http://www.atlassian.com/software/jira/
[5] http://www.atlassian.com/software/fisheye/

Abschluss in den Head überführt. Dabei muss folgende Tag-Konvention strikt eingehalten werden:

OLAT-[version|head]-[urheber]-[jira-issue-nummer]
: Somit würde zum Beispiel OLAT-HEAD-BPS_UZH-1509 für einen Branch mit einem vom Bildungsportal Sachsen GmbH und der Universität Zürich zu entwickelnden Feature stehen, welches sich auf die Jira-Issue Nummer 1509 bezieht. Außerdem ist hier noch darauf hinzuweisen, dass für die Einhaltung der Tag-Konvention mindestens ein Jira-Issue für die Eröffnung eines entsprechenden Branches vorhanden sein muss.

6.2 Konfigurationsmanagement der Abteilung Betriebliche Informationssysteme

Auf Grund der aktuellen und auch zukünftigen Vielfalt der in der Abteilung Betriebliche Informationssysteme entwickelten Erweiterungen für das Learning Management System OLAT muss eine einheitliche und wohldefinierte Vorgehensweise zur Identifikation von Konfigurationseinheiten gefunden werden. Die wohl bedeutsamste Einheit in diesem Zusammenhang ist das BIS-OLAT. Diese Einheit wird definiert als eine Zusammenfassung aller sinnvollen und kompatiblen Erweiterungen zu einer gegebenen OLAT-Version. Das heißt, dass hier alle Erweiterungen integriert werden, die neue Funktionalität in ein vorhandenes System integrieren und sich hierbei nicht gegenseitig im Konflikt stehen. Beispielsweise wäre eine Verknüpfung der Funktionalität von XMAN mit der von Olate im praktischen Einsatz durchaus interessant, wohingegen die durch das ADAG-Projekt bereitgestellte Applikation eine völlig von den bisherigen Anforderungen abweichende Anwendung darstellt.
Alle oben genannten sowie noch folgende Erweiterungen werden ebenfalls als Konfigurationseinheiten betrachtet und somit in der Konfigurationsdatenbank verwaltet. Abbildung 6.1 gibt einen Überblick über eine mögliche Ausprägung dieser Datenbank.
Somit ist festzuhalten, dass jede entwickelte Erweiterung einen Eintrag im Head des SVN-Repositories erhält. Außerdem muss sichergestellt sein, dass bei einem Checkout eines Projektes aus dem Head respektive Trunk eine vollständige und funktionstüchtige Einheit heruntergeladen wird. Folglich enthalten diese Konfigurationseinheiten den von OLAT selbst sowie den eigens implementierten Quellcode. Extensions, die für eine Migration in das BIS-OLAT Projekt vorgesehen sind, werden in Branches entwickelt. Dort lagern allerdings nur die Klassen, die tatsächlich vom hiesigen Entwicklerteam erzeugt wurden. Sind

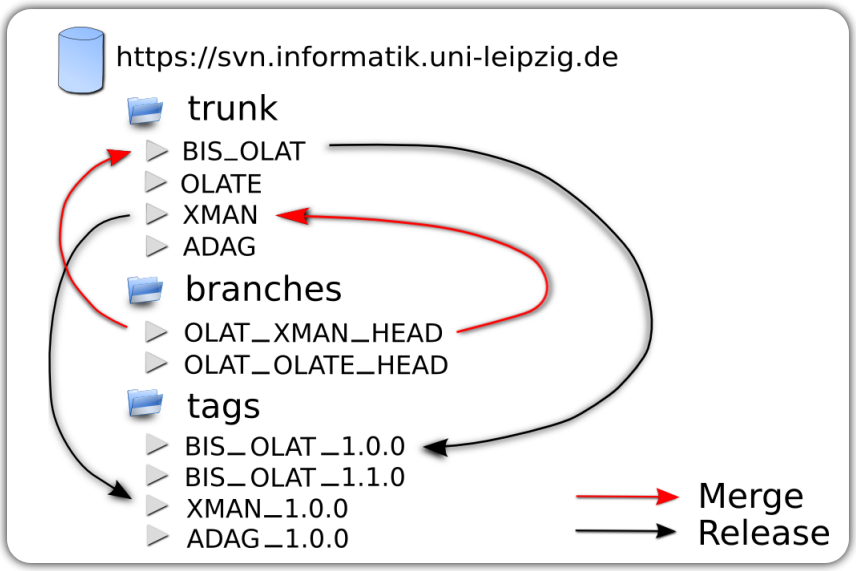


Abbildung 6.1: Mögliche Ausprägung der Konfigurationsdatenbank

die Arbeiten an den, in den Entwicklungszweigen vorliegenden, Projekten abgeschlossen oder erreichen zumindest ein neues Bugfix Release, werden die Änderungen in das eigenständige Projekt im Trunk und in das BIS-OLAT Projekt überführt. Dieser Vorgang wird auch als Merging bezeichnet und wird im Wiki[6] der Technischen Universität Chemnitz erläutert. Die aus der Weiterentwicklung der einzelnen Features entstandenen Versionen werden im Tags Ordner des SVN-Repositories verwaltet. Einen beispielhaften zeitlichen Überblick über eine mögliche Ausprägung dieser Konfigurationsdatenbank mit zwei Erweiterungen wird in Abbildung 6.2 angeführt.
Einen weiteren Bestandteil der Projektkonfiguration bildet die Bug-Tracking-Software Bugzilla[7]. Für jedes der im Trunk vorhandenen Projekte, also BIS-OLAT, XMAN und andere, wird eine solche Instanz aufgesetzt. Falls es möglich sein sollte, in einer Bugzilla Instanz mehrere Projekte gleichzeitig zu verwalten, wird diese Variante präferiert. Es ist ebenfalls darauf zu achten, dass jeder Commit-Vorgang der Codebasis einer entsprechenden Issue-Nummer zugeordnet sein muss. Ist für eine Veränderung des Quellcodes keine zugehörige Fehlernummer in der Bugzilla-Instanz vorhanden, muss eine solche generiert werden. Dadurch kann eine direkte und vor allem ständig nachvollziehbare Verbindung zwischen dem Fehler sowie dessen Begleitumstände und der entsprechenden Klasse respektive der Methode, der Applikation erzeugt und dokumentiert werden.

[6] https://twiki.tu-chemnitz.de/bin/view/OLAT/HowToFeatureBranch
[7] http://www.bugzilla.org/

Um die an der Fakultät entwickelten Projekte der Öffentlichkeit präsentieren zu können, sollte zudem ein eigenständiges Wiki aufgesetzt werden. Hier sollte über den Fortschritt der einzelnen Projekte berichtet, Installationsanweisungen bereitgestellt, Arbeiten rund um OLAT veröffentlicht und mindestens das neueste Release zum Herunterladen angeboten werden.

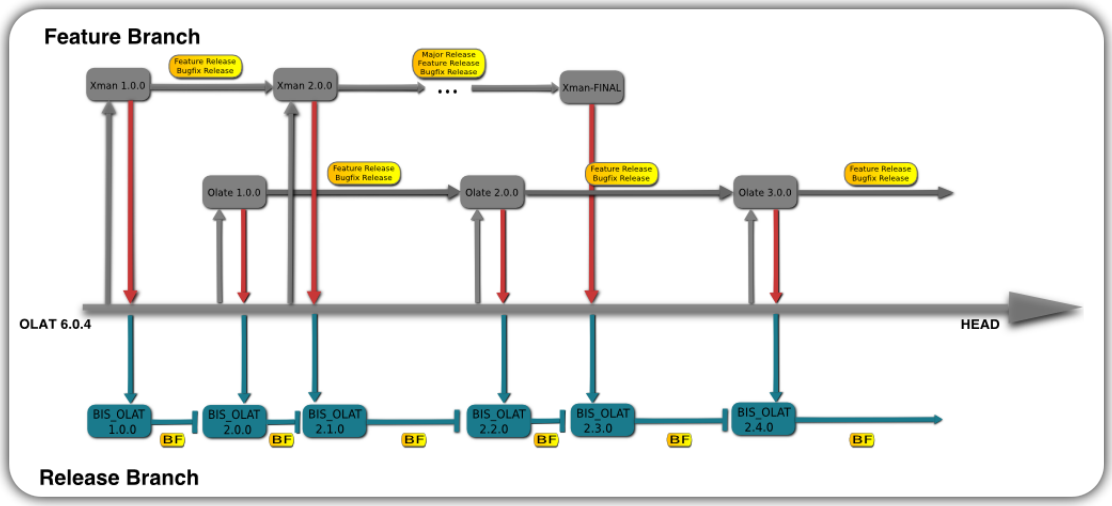


Abbildung 6.2: Branching der BIS-OLAT-Erweiterungen

6.3 Versionsmanagement

Die Versionierung erfolgt grundsätzlich durch Versionsnummern. Dabei wird nicht zwischen Versionen des BIS-OLAT Projekts und den einzeln Entwicklungszweigen unterschieden, das heißt Branches folgen dem gleichen Versionierungsschema wie die im Head entwickelten Projekte. Das folgende Schema beschreibt die Versionsnummer eines Releases, also eine an einen Kunden auszuliefernde Version:

[RELEASE-NAME] - [M.F.B] - [YYYYMMDD]

Der Name und die Bedeutung eines Releases wird im Kapitel 6.4 näher erläutert. Die Versionsnummer setzt sich aus drei natürlichen Zahlen und dem Veröffentlichungsdatum zusammen, deren Semantik folgendermaßen definiert ist:

Major:

Die erste Stelle der Versionsnummer wird nur erhöht, wenn fundamentale Änderun-

gen beziehungsweise Neuerungen in den Quellcode des Releases aufgenommen wurden. Dies würde beispielsweise für das BIS-OLAT Projekt bedeuten, dass die erste bedeutende Version, eines in einem Entwicklungszweig fertiggestellten Features zur Verfügung steht. Für diese entwickelten Features wird die erste Stelle zum Beispiel erhöht, wenn ein Feature der Größenordnung der elektronischen Studentenakte zum bestehenden Prüfungsverwaltungssystem hinzugefügt wurde.

Feature:

Wie bisherige Erfahrungen gezeigt haben, entstehen nach der Veröffentlichung einer neuen Version bei den Nutzergruppen fortlaufend neue Anforderungen an das System. Dazu gehören beispielsweise Erweiterungen an vorhandenen Tabellen oder das Hinzufügen von etwaigen Kommunikationsmöglichkeiten zwischen Prüfer und Prüfling. Um über diese stetigen und kleineren Veränderungen einen Überblick zu behalten, sollten in einem regelmäßigen Abstand neue Feature-Releases veröffentlich werden. Dies bezieht sich zur Zeit hauptsächlich auf die in den Branches entwickelten Features, da die Weiterentwicklung von OLAT in den Händen der Züricher Entwickler liegt.

Bugfix:

Da Major- sowie Feature-Releases voraussichtlich niemals fehlerfrei veröffentlicht werden, sind Bugfix-Release von Nöten. Es ist darauf zu achten, dass ausschließlich kleinere Veränderungen der Codebasis, die zur Beseitigung von auftretenden Fehlern notwendig sind, in diesen Veröffentlichungen erscheinen.

YYYYMMDD:

Das Veröffentlichungsdatum wird zum Beispiel wie folgt notiert: 20091126, was auf einen Termin der Veröffentlich am 26. November 2009 hinweist.

6.4 Releasemanagement

Durch den Open-Source Charakter der zu entwickelnden OLAT-Erweiterungen ergeben sich zwei differierende Kundengruppen. Um beiden Gruppierungen gerecht zu werden, muss zwischen einem Release für den Produktiveinsatz für die Anwendergruppe und einem Release für die technische Weiterentwicklung beziehungsweise Anpassung für die Gruppe der Softwareingenieure differenziert werden. Das Hauptaugenmerk soll hier nicht unbedingt auf dem Inhalt der ausgelieferten Version liegen, sondern vielmehr auf dem verwendeten Vertriebsweg einerseits und der Komplexität des Installationsprozesses andererseits.

Es ist hier besonders darauf zu achten, dass ein der Anwendergruppe zur Verfügung gestelltes Release eine möglichst ausführliche und leicht verständliche Installationsanleitung enthält, sowie dessen Installation selbst auch von Nichtfachleuten durchgeführt werden kann. Diese Einschränkungen gelten erfahrungsgemäß nicht für Fachleute, womit hier der Distributionsweg direkt über einen anonymen und lesenden SVN-Zugang angeboten werden sollte. Außerdem muss darauf geachtet werden, dass hier zwischen einer Neuinstallation, einer Aktualisierung oder der Integration einer Erweiterung in bereits bestehende Produktivsysteme unterschieden wird.
Überdies muss für einen einheitlichen Releasemechanismus der Inhalt der einzelnen Releases definiert werden. Infolgedessen besteht ein Release nicht nur aus dem benötigten Quellcode, sondern auch aus einer vollständigen Menge von Installationsskripten, sowohl Skripten für die mögliche Änderung des Datenbankschemas als auch das eigentliche Applikationsinstallationsskript, veränderten oder hinzugefügten Konfigurationsdateien und Bildern wie auch der Dokumentation in Form des Handbuchs und der Quellcodedokumentation im HTML-Javadoc-Format. Überdies sollte jedem Release eine für den Endanwender verständliche Liste an Verbesserungen zur Vorgängerversion beiliegen.

6.5 Änderungsmanagement

Im Laufe des Lebenszyklus einer für OLAT entwickelten Erweiterung werden Änderungen der fakultätsspezifischen Bedürfnisse und Anforderungen auftreten. Das heißt, es werden Ansprüche an die Applikation gestellt, die bei der Entwicklung und Planung des zugrunde liegenden Modells nicht beachtet wurden, aber einen großen Beitrag zur Optimierung der Prozesse an der Fakultät beisteuern könnten. Dieser Faktor würde bei einer zukünftigen intensiveren Nutzung, auch durch andere Abteilungen des Instituts für Informatik, noch vielfach verstärkt werden. Um diese Änderungsvorgänge möglichst präzise und vollständig strukturieren und dokumentieren zu können, wird in den folgenden Abschnitten ein Konzept vorgestellt, das dies zum Ziel hat.

6.5.1 Funktionale Erweiterung und Fehlerbehebung

Wie in Abschnitt 6.2 bereits erwähnt, werden Änderungswünsche in einer projektspezifischen Bugzilla-Instanz organisiert. Zu den dort verwalteten Einträgen gehören neben neuen Produktfunktionen auch die durch die Entwickler- und Nutzergruppe festgestellten Mängel des Systems. Zur vollständigen Dokumentation ist es hier zwingend erforderlich,

einen Verantwortlichen für diese Änderung festzulegen. Dies bringt vor allem bei einem sich ablösenden Entwicklerteam den Vorteil, dass die für ein bestimmtes Problem zuständige Person direkt ermittelt und befragt werden kann. Außerdem sollten ebenfalls weitere Informationen wie Priorität und Art[8] des Eintrags fixiert werden.
Die durch die Bugzilla-Einträge repräsentierten Modifikationen der Applikationen werden der Konfigurationsdatenbank nach dem folgenden Muster übergeben. Jeder Übergabevorgang, im Folgenden Commit genannt, bezieht sich nur auf eine einzelne Klasse. Das heißt, es wird für jede Klasse einzeln ein Commit mit einer Reihe von Metainformationen durchgeführt. Diese Informationen werden, wie auch die Informationen in der Bugzilla-Instanz selbst, in englischer Sprache formuliert und zusätzlich in einer projektbegleitenden Datei gespeichert. Im Detail handelt es sich hierbei um:

1. Bugzilla-Issue-Nummer |
2. Referenz zu einer Bugzilla-Issue-Nummer (optional) |
3. Wann wurde die Änderung vorgenommen? |
4. Wer hat die Änderung vorgenommen? |
5. Welche Klasse respektive welche Methoden sind betroffen? |
6. Warum wurde die Änderung durchgeführt?

Ein Beispiel könnte wie folgt aussehen:

> 0001 | 0000 | 17.03.2009 | Daniel Gerber | CreateAndEditAppointmentForm.validate(): date gets checked if it's a correct one | entering of 36.04.2009 was possible and caused an Exception

Ferner sollte dafür gesorgt werden, dass die an der Entwicklung von Erweiterungen für das OLAT-System beteiligten Personen über ein Verständnis aller gegenwärtigen, in den Entwicklungszweigen befindlichen, Arbeiten besitzen. In Folge dessen können Konflikte und eventuell auftretende Integrationsprobleme möglicherweise frühzeitig erkannt und behoben werden.
Weiterhin sollte auf der Wiki-Projekt-Seite eine Roadmap, also ein Plan über zukünftige Entwicklungen, veröffentlicht werden und Nutzern eine Möglichkeit gegeben werden, Feedback an die Entwickler zu senden. Dies könnte beispielsweise durch eine Mailingliste oder ein Forum bewerkstelligt werden.

[8] Bugreport eines Entwicklers, Bugreport eines Nutzers, Wunsch für Review eines Codefragments

6.5.2 Upgrade-Mechanismus auf neuere OLAT-Versionen

Eine der wohl schwierigsten Aufgaben der Planung des Änderungsmanagements ist es, einen wohldefinierten Prozess zu finden, um bestehende Erweiterungen an eine neue OLAT-Version anzupassen. Dies ist im Allgemeinen eine sehr zeitkomplexe Aufgabe. Erschwerend hinzu kommen noch die geringen zur Verfügung stehenden personellen Kapazitäten, sodass aus diesem Grund nicht für jedes neue Feature-Release von OLAT, dieser Vorgang durchgeführt werden kann. Dennoch muss sichergestellt sein, dass dieser Vorgang regelmäßig stattfindet, um den technischen Anschluss nicht zu verlieren.
Dieser Prozess könnte in Zukunft deutlich vereinfacht werden, wenn wie angekündigt in der OLAT Version 6.2 das Rollenmanagement überarbeitet und ein einheitlicher Erweiterungspunkt für Lernressourcen geschaffen wird. Momentan ist es beispielsweise in der Erweiterung XMAN nötig, Quelldateien von OLAT zu verändern. Dieser Schritt würde folglich wegfallen und die Konfiguration von Erweiterungen in einer Reihe von XML-Dateien stattfinden.

Mechanismus in kommerziellen Unternehmen

Um ein angemessenes Konzept für den Update-Vorgang von bisher entwickelten Erweiterungen zu finden, wird zuerst das Update-Management eines kommerziellen Unternehmens, in diesem Fall der Frentix GmbH[9], analysiert. Aus den hierbei resultierenden Ergebnissen wird ein Prozess für die in der Abteilung für Betriebliche Informationssysteme entstandenen Modifikationen abgeleitet.

Frentix ist als Spin-Off Unternehmen aus dem an der Universität Zürich entwickelten Open-Source Projekt OLAT hervorgegangen. Seitdem verfolgt es dessen Verbreitung sowie Weiterentwicklung und bewegt sich in den Tätigkeitsbereichen E-Learning, Softwareentwicklung, Multimedia, Fotographie und Medienproduktion. Durch die Unternehmensgeschichte wird Frentix ermöglicht, einen großen Teil der Anforderungen ihrer Kunden direkt in OLAT zu integrieren. Das heißt, die von Frentix entwickelten Erweiterungen wandern direkt in den Head des CVS-Repositories. Zusätzlich wird darauf geachtet, dass möglichst alle Änderungen durch Konfigurationsdateien oder existierende Erweiterungspunkte realisiert werden können. Hieraus ergibt sich, dass nur wenige Modifikationen an den OLAT Klassen selbst durchgeführt werden müssen. Hierfür hat Frentix den in Abbildung 6.3 dargestellten Arbeitsablauf definiert.
Sobald für ein neues, von der OLAT Projektgruppe freigegebenes Release ein Release-Branch eröffnet wird, beispielsweise der 6.1-Branch, wird eine Patch Datei generiert, die

[9] http://www.frentix.com

alle Unterschiede zwischen dem Frentix 6.0-Branch und dem OLAT 6.0-Branch enthält. Des Weiteren wird vom OLAT 6.1-Branch der Frentix 6.1-Branch abgezweigt und in das frentix-interne CVS-Repository integriert. Die im ersten Schritt generierte Patch Datei wird nun auf den Frentix 6.1-Branch angewendet. Allerdings müssen nun alle hierbei auftretenden Konflikte manuell aufgelöst werden. Abschließend wird der neue Branch wie in Abschnitt 5.1 demonstriert, auf noch vorhandene Fehler untersucht und gegebenenfalls eine Korrektur vorgenommen.

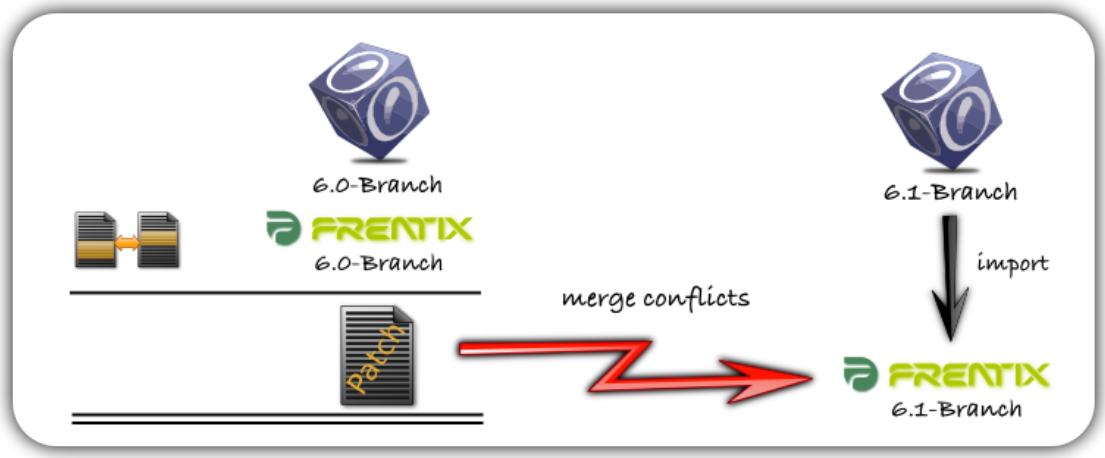


Abbildung 6.3: Updatemechanismus der Frentix GmbH

Mechanismus an der Universität Leipzig

Wie bereits in Abschnitt 6.5.2 erwähnt, muss ein regelmäßiger Updatevorgang auf neuere OLAT Versionen stattfinden. Hierzu wird vorgeschlagen, diesen Prozess nach dem Ablauf einer dreimonatigen Frist nach Erscheinen eines neuen Feature Release von OLAT anzustoßen. Der Grund für diese Verzögerung ist, dass auf ein Feature Release in der Regel eine Reihe von kleineren Bugfix Releases folgen, welche die Stabilität erhöhen und die Anzahl der enthaltenen Fehler der jeweiligen Version stark verringern. Somit bildet die Grundlage für das Update das aktuellste Bugfix Release, welches drei Monate nach dem zugehörigen Feature Release zur Verfügung steht. Dadurch wird garantiert, dass für die Dauer von mindestens sechs Monaten, also dem Zeitraum zwischen zwei OLAT Feature Releases, ein stabiles Grundgerüst, für die in der Abteilung Betriebliche Informationssysteme durchzuführenden Entwicklungsaufgaben, bereit steht. Unglücklicherweise ist der von Frentix vorgeschlagene Patch-Mechanismus für Erweiterungen, wie zum Beispiel XMAN, ungeeig-

net, da hier eine Vielzahl von OLAT Klassen geändert werden musste. Somit würde durch dieses Patching wesentlich mehr Arbeitsaufwand entstehen. Der dazu vorgeschlagene und alternative Arbeitsablauf wird in 6.4 Abbildung dargestellt, sowie folgendermaßen definiert.
Der erste Arbeitsschritt, nach Ablauf der dreimonatigen Wartefrist, ist das Importieren der aktuellsten OLAT Version in den Trunk der Konfigurationsdatenbank. Dies geschieht entweder durch die von CVS gegebene Vendor Branch Import Funktion oder wird von Hand durchgeführt. Anschließend wird diese Version von jedem beteiligten Entwickler einmalig in den persönlichen Eclipse Workspace integriert. Nun werden für alle in der momentan aktuellsten Version der jeweiligen Erweiterung vorliegenden Klassen sämtliche korrespondierenden Klassen des neuen OLAT Projekts, entsprechend der vorhandenen Ordnerstruktur, in ein neues Projekt aufgenommen. Diese Paare von beinahe identischen Klassen werden nun über die von Eclipse bereitgestellte Funktion *Compare With > Each Other* miteinander verglichen und nach bestem Wissen zusammengeführt. Ist dieser Mergevorgang beendet, werden alle restlichen Klassen und Dateien in das neue Projekt übertragen. Sollten nun trotz manueller Übernahme der entsprechenden Änderungen Fehler im neuen Projekt vorhanden sein, müssen diese bevor weitere Schritte erfolgen können, gelöst werden. Hierzu könnte es durchaus nötig sein, umfangreiche Recherchen durchzuführen. Somit sollte für diesen Schritt ein größeres Zeitfenster eingeräumt werden. Ist dieser Vorgang abgeschlossen, wird eine ausführliche JUnit Test Serie gestartet. Ausführliche Hinweise hierzu, befinden sich im Kapitel 5. Diese Testrunden werden solange iteriert, bis die aktualisierte Erweiterung keine bekannten Fehler mehr enthält. Die letzte Stufe dieses Prozesses ist das Importieren der nun aktualisierten OLAT-Erweiterung in den Trunk beziehungsweise die Branches der Konfigurationsdatenbank.

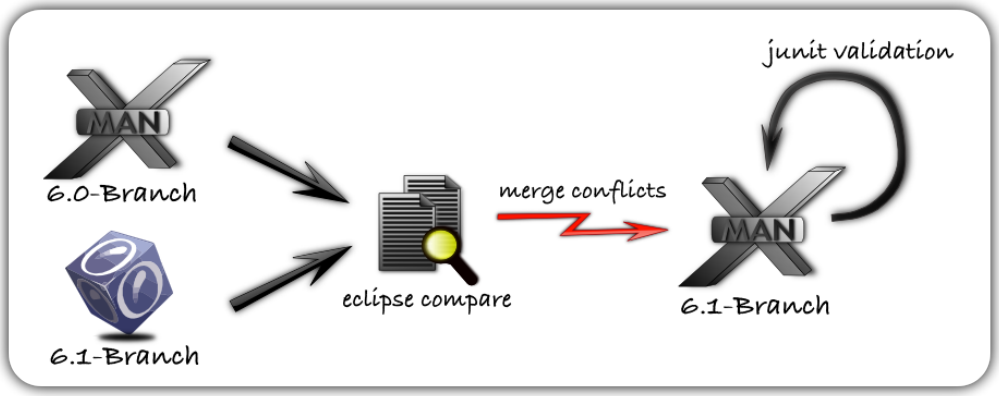


Abbildung 6.4: Updatemechanismus der Xman Erweiterung

7 Evaluation der Prüfungsverwaltungssoftware XMAN

7.1 Ziele der Evaluation

In der Prüfungsperiode des Sommersemester 2008 wurde am Lehrstuhl für Softwaretechnik erstmals die Prüfungsverwaltungssoftware "XMAN" eingesetzt. Dieser Einsatz erfolgte mit der zu der Zeit aktuellsten Version 0.1-Alpha. Da für die Entwicklung dieser Version ein sehr kurzer Zeithorizont vorgesehen war und des Weiteren keine von offizieller Stelle bestätigten Requirements vorlagen, kann in erster Linie nicht davon ausgegangen werden, dass die bereitgestellte Funktionalität den Anforderungen der Benutzer genügt. In diesem Zusammenhang muss sich eine Bewertung des Gesamtsystem aus der Summe der Bewertungen der jeweiligen Einschätzungen der drei relevanten Benutzergruppen zusammensetzen. Zu diesen Gruppen zählen:

1. Studenten der Universität - Rolle in XMAN: User
2. Prüfungsamt der Fakultät - Rolle in XMAN: ExamAdmin
3. Professoren der Fakultät - Rolle in XMAN: Autor

Auf Grund der verschiedenartigen Anforderungen der einzelnen Benutzergruppen an das System muss das primäre Ziel der Evaluation die Strukturierung und Auswertung der Erfahrungen der unterschiedlichen Nutzer sein. Für die zufrieden stellende Bearbeitung dieser Aufgabe wird die Evaluation in drei Teile gegliedert.

7.1.1 Befragung der Studenten

Den Großteil der am System angemeldeten Nutzer bilden Studenten. Somit ist die Anzahl an gesammelten Erfahrungen in dieser Benutzergruppe am größten und durch die wahrscheinlich große Meinungsdiversität auch am schwierigsten zu analysieren. Es muss hier also ein besonderer Wert auf möglichst präzise und unmissverständliche Fragestellungen gelegt werden. Die Evaluation der studentischen Erfahrungen zielt auf eine möglichst

genaue Einschätzung des Verhaltens und der Benutzeroberfläche der Prüfungsverwaltungssoftware. Genauer soll untersucht werden, ob die Studenten mit den für sie bereitgestellten Funktionen zufrieden sind und umgehen können. Die so erhaltenen Informationen sollen somit als Grundlage für zukünftige Implementierungsarbeiten dienen und zu einer deutlichen Verbesserung der "Usability" aus Sicht der Studenten führen. Außerdem soll durch das ermöglichte Feedback die Akzeptanz für die PA-OLAT Instanz gestärkt und ausgebaut werden, sowie das System nutzerfreundlicher gestaltet werden.

7.1.2 Befragung des Prüfungsamtes

Ein von XMAN verfolgtes Ziel ist die einfache Übergabe der Prüfungsergebnisse von Prüfenden an das Prüfungsamt. Die Evaluation soll hier zeigen, in wie weit dieser Informationsaustausch beiden Seiten von Nutzen ist und die fakultätsinterne Kommunikation entlastet. Des Weiteren soll die Funktionalität des elektronischen Äquivalents der bis heute physisch gespeicherten Studentenakte evaluiert werden. Da aktuell nur Herr Reutter als Mitglied des Prüfungsamts an XMAN arbeitet, wird hier ein leitfadengesteuertes Interview durchgeführt.

7.1.3 Befragung der Professoren

Das primäre Ziel von XMAN ist nicht die vollständige Abbildung der Aktivitäten der Prüfungszeit, sondern die Unterstützung der zugrunde liegenden administrativen Prozesse der Fakultät und Mitarbeiter. Um diese Unterstützung auf einem möglichst hohen und professionellen Level zu gewährleisten, werden die Erfahrungen der Prüfer evaluiert und mit deren Hilfe die bereitstehenden Prozesse korrigiert und optimiert. Dies geschieht allerdings nicht im Rahmen eines Interviews oder Fragebogens, sondern wurde bereits während des gesamten Semesters durch den stetigen Austausch von Erfahrungen und Funktionswünschen zwischen Entwickler und Anwender abgedeckt.
Der Einsatz der Prüfungsverwaltungssoftware beschränkte sich in der ersten Testphase ausschließlich auf Prüfungen des Lehrstuhls für Softwaretechnik. Um zukünftig ein breiteres Einsatzgebiet zu erlangen sollen die gesammelten Erfahrungen der Prüfer helfen, bestehende Missstände aufzudecken und zu beseitigen.

7.2 Methodik für Studentenbefragung

Um nachzuweisen, ob die XMAN-Software im Stande ist, die benötigte Funktionalität zu erbringen und einen tatsächlichen Vorteil zur Vorgängerlösung sicherstellt, wird der Be-

fragungstyp Fragebogen ausgewählt. Die Gründe für diese Wahl liegen in der schnellen und unkomplizierten Verbreitung des Fragebogens an die beteiligten Personen, sowie in der einfachen Auswertung der Ergebnisse. Außerdem kann hier auf die schon von OLAT vorgegebene Lernressource Fragebogen zurückgegriffen werden. Die Fragen werden in den meisten Fällen geschlossen formuliert. Das heißt, der Befragte hat die Möglichkeit, aus einer Reihe von vorgegebenen Antworten die für sich am besten passende auszuwählen. Dieser Fragetyp wird zum Beispiel bei der Bewertung der Handhabung und Optik der Software eingesetzt. Teilweise werden auch Fragen offen formuliert. Das bedeutet, es werden keine Antwortmöglichkeiten vorgegeben, so dass der Befragte seine Antwort vollständig selbst bilden muss. Dies tritt beispielsweise bei der Frage nach gewünschten Verbesserungsvorschlägen auf. Hier soll dem Antwortenden keine mögliche Verbesserung suggeriert werden, sondern dessen Phantasie und Kreativität angeregt werden. In seltenen Fällen tritt auch eine Mischform aus diesen beiden Fragetypen auf.

7.3 Durchführung der Studentenbefragung

Die Evaluation wurde mit Hilfe der von OLAT bereitgestellten Lernressource Fragebogen durchgeführt. Allen Nutzern der OLAT-PA Plattform wurde per OLAT-interner Mail-Funktion ein Link hierzu gesendet. Die Ergebnisse wurden dann von OLAT anonymisiert exportiert. Das heißt jeder Befragte erhielt eine unpersönliche Laufnummer. Der Export dieser Resultate erfolgte im xls-Format und wurde anschließend graphisch aufbereitet und inhaltlich ausgewertet. Die ausführliche Auswertung der Evaluation erfolgt im nächsten Abschnitt.

7.4 Auswertung

Im folgenden werden die Ergebnisse der Evaluation der Prüfungsverwaltungssoftware gegenüber den Nutzergruppen Studenten und Prüfungsamtmitarbeiter dargestellt. Dabei ist zu beachten, dass auf Grund der Menge und Vielfalt an Fragen nur ein spezieller, für die Bewertung der Software notwendiger, Teil ausgewertet wurde.

7.4.1 Prüfungsamt

Durch das Interview mit Herrn Reutter ergab sich, dass das Prüfungsamt zum Zeitpunkt der Evaluation in nur sehr geringem Maße mit der Prüfungsverwaltungssoftware gearbeitet hat. Insbesondere wurden Prüfungsergebnisse nicht über die XMAN-Funktion wei-

tergeleitet, wodurch eine Evaluation hier unmöglich wird. Somit bestand auch nicht die Notwendigkeit, das Handbuch zu lesen, was ebenfalls eine Auswertung verhindert. Die bis zu diesem Zeitpunkt verwendeten Funktionen wie Validierung der Studentenakten, das Anlegen und Verwalten von Studiengängen und Module verlief allerdings fehlerfrei. Zur Verbesserung der operativen Unterstützung wurde eine automatische Emailbenachrichtigung vorgeschlagen, die eine xls-Datei mindestens bestehend aus Matrikelnummer, Note und Studiengang bei erfolgter Benotung der Prüfung an das Prüfungsamt sendet.

7.4.2 Studenten

An der Evaluation der studentischen Nutzergruppe haben insgesamt 34 Personen teilgenommen. Darunter waren 30 männliche und vier weibliche Teilnehmer. Das durchschnittliche Alter der Probanden liegt bei 23 bis 25 Jahren, die Fachsemester bewegen sich zwischen dem 1. bis 4. Fachsemester und dem 5. bis 11. Fachsemester relativ gleichmäßig. Der am meisten belegte Studiengang ist Bachelor Informatik (neu) gefolgt von Master Informatik (neu) und Diplom Informatik. Die größte Anzahl der Nutzer ist erst ungefähr 1-5 Monate mit OLAT vertraut und hat außerdem erst eine Prüfung mit Hilfe von XMAN absolviert.

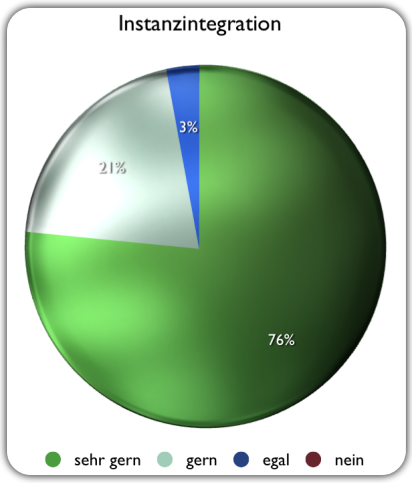


Abbildung 7.1: Evaluation des Integrationswunschs

Wie Abbildung 7.1 zeigt, kommt es bei der Frage nach einer einheitlichen Lösung zur Prüfungsverwaltung wie erwartet zu einer eindeutigen Antwort. Insgesamt 97% sprechen sich für eine zentrale Stelle zur Prüfungseinschreibung aus. Durch die Antworten der Testpersonen auf die offene Frage wurde hier außerdem deutlich, dass den meisten Nutzern die veraltete Anmeldung über das Prüfungsamt ausreicht, sie aber auch mit einer zentralen

Registration aller Prüfungen über XMAN einverstanden wären. Wenn auch die Anforderungen des Prüfungsausschusses die strikte Trennung zwischen Instanzen zur Lehrveranstaltungsorganisation und Prüfungsdurchführung einzuhalten besteht, sollte dennoch überlegt werden, wie man diese beiden Instanzen enger miteinander verbinden könnte.
Ein weiterer Bereich des Fragebogens betrifft die Einschätzung der drei Themengebiete Handbuch, Prüfungstermine im XMAN-Kalender und Emailbenachrichtigung. Da zum Zeitpunkt der Evaluation keiner der Probanden das Handbuch gelesen hatte, kann hier keine Einschätzung über die Qualität des Dokuments getroffen werden. Auch Kalendereinträge werden von der Nutzergruppe nur spärlich genutzt. Abbildung 7.2 zeigt, dass mit 19 Studenten über die Hälfte der Nutzer die Kalenderfunktion nicht nutzen. Dies ist vermutlich der Tatsache geschuldet, dass die an der Evaluation teilnehmenden Studenten lediglich eine Prüfung über XMAN absolvierten und dafür keine Kalenderfunktion benötigen. Ein interessanter Vorschlag eines Teilnehmers zur Verbesserung dieser Situation wäre die Publikation eines öffentlichen Kalenders mit allen Prüfungsterminen im iCal-Format[1].
Die Emailbenachrichtigung wurde von 64% mit gut und besser bewertet. Außerdem konnten durch die Evaluation auch Fehler des Inhalt der Emails ausgebessert werden.

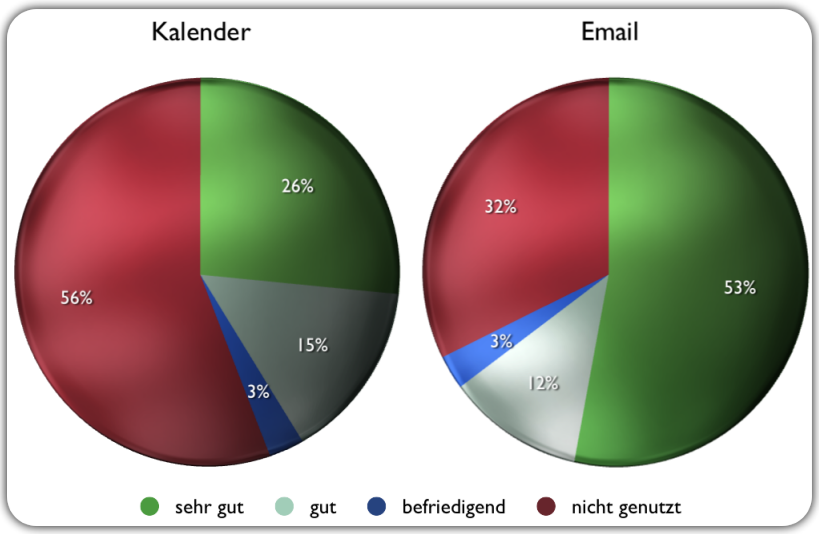


Abbildung 7.2: Evaluation der Kalender- und Emailkomponente

Der Schwerpunkt der Evaluation lag auf der Einschätzung der Abbildung und Integration der Prüfungen sowie die Gestaltung der elektronischen Studentenakte. Dazu wurden

[1] http://tools.ietf.org/html/rfc2445

die Testpersonen gefragt, wie sie die Beantragung der ESA, den Inhalt und die Funktionalität der Akte respektive der Prüfungen beurteilen. Die Ergebnisse dieser Fragen werden in Abbildung 7.4 und 7.3 dargestellt.

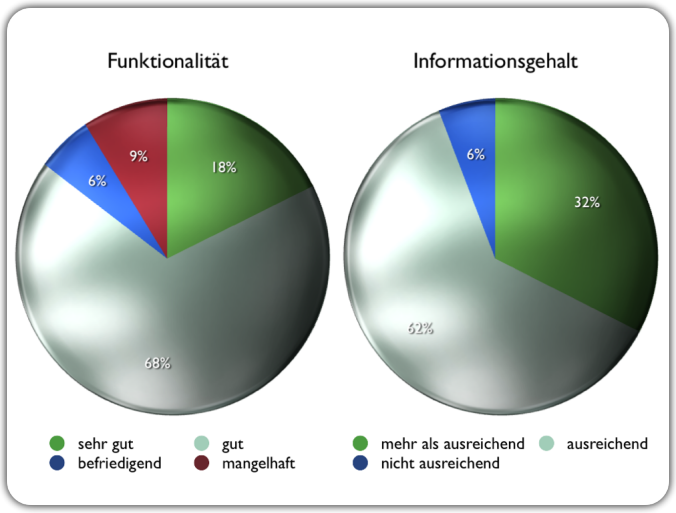


Abbildung 7.3: Evaluation der Prüfung

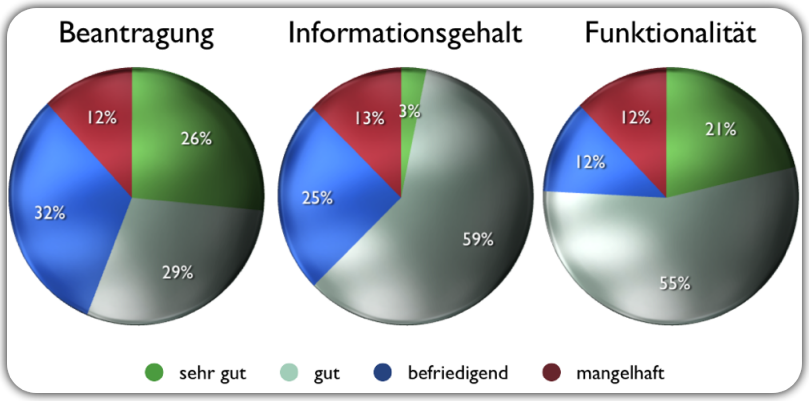


Abbildung 7.4: Evaluation der elektronischen Studentenakte

Dabei zeigt die Abbildung 7.3 deutlich, dass über 80% der Befragten mit der Funktionalität und dem Informationsgehalt der Prüfungen in XMAN zufrieden oder mehr als zufrieden sind. Der optische Eindruck (hier nicht graphisch dargestellt) wird allerdings von circa 33% mit gefällt mir weniger oder gefällt mir gar nicht bewertet. Dies könnte allerdings von der allgemeinen Unzufriedenheit mit der OLAT-Technologie selbst herrühren. So wurden beispielsweise mehrmalig Kommentare über die fehlende 'Zurück-Funktionalität'

sowie 'Reiter-Funktionalität' gegeben. Die elektronische Studentenakte schneidet im Hinblick auf Informationsgehalt und Funktionalität ähnlich erfolgreich ab wie die Prüfungen. Der optische Eindruck (hier ebenfalls nicht graphisch dargestellt) fällt aber noch unter die Werte der Prüfungen. Hier bewerten circa 50% der Teilnehmer die Oberfläche als an manchen Stellen verwirrend beziehungsweise überhaupt nicht den eigenen Vorstellungen entsprechend. Der oben genannte Grund tritt in diesem Fall sicherlich wieder ein, allerdings sollte an dieser Stelle eine Überarbeitung der Oberfläche stattfinden. Auch die Frage nach der Beantragung der elektronischen Studentenakte bewerten 45% der Befragten mit befriedigend und mangelhaft. Dies ist allerdings, wie sich während der Prüfungsperiode herausgestellt hat, dem Unvermögen einiger Nutzer geschuldet, die Instruktionen der XMAN-Startseite nicht oder nicht korrekt zu lesen. Außerdem sollte die in Abschnitt 4.7.1 eingeführte LDAP-Authentifikation eine Verbesserung dieser Ergebnisse mit sich bringen.

8 Zusammenfassung

Die vorliegende Arbeit befasste sich mit der Fragestellung, wie etablierte Prozesse des Prüfungsmanagements einer universitären Fakultät durch Software abgebildet werden können, um eine Senkung des Organisationsaufwandes für die daran beteiligten Personen zu erzielen. Dabei diente das bereits zur Verwaltung von Lehrveranstaltungen eingesetzte Learning Management System OLAT als Grundlage für diese Arbeit. Ausgehend von den speziellen Anforderungen der Fakultät für Informatik der Universität Leipzig wurde eine Softwarearchitektur vorgestellt, die sich tief in die bereits etablierte Codebasis des Open Source Projekts OLAT integriert. Dabei wurde darauf geachtet, dass in OLAT vorhandene Konventionen und Richtlinien zur Geschäftsprozessmodellierung strikt eingehalten wurden und das entwickelte System aus einer Reihe möglichst unabhängiger Komponenten besteht. So wurde beispielsweise die Prüfung und die elektronische Studentenakte als größte entwickelten Komponenten detailliert vorgestellt. Außerdem wurde ein Ausblick auf mögliche zukünftige Erweiterungen des PVS gegeben.

Der zweite große, eher theoretische Teil dieser Arbeit beschäftigte sich mit Fragen des Softwarequalitätsmanagements und dem Konfigurationsmanagement. So wurde in beiden Fällen eine Analyse der von OLAT vorgegebenen Managementprozesse durchgeführt und darauf aufbauend ein für weitere Arbeiten zu Grunde liegendes Testkonzept respektive Konfigurationsmanagement entwickelt. Das Testkonzept liefert dabei, neben einer Standardisierung von Testfällen, Richtlinien zum Testen von Objekten verschiedenster Größe. Es wurde das Testen von einzelnen Klassen ebenso behandelt, wie die Einrichtung eines entwicklungsbegleitenden Testsystems, die Durchführung von Komponententests auf Basis von Geschäftsprozessen oder die Durchführung von regelmäßigen Durchsprachen. Im Kontext des Konfigurationsmanagements wurden Managementprozesse definiert, die den Umgang mit der Versionierung und dem Veröffentlichen der entwickelten Software regeln. Außerdem wurde eine ausführliche Richtlinie für das in diesem Zusammenhang komplexeste Gebiet des Upgrade auf neuere OLAT Versionen präsentiert.

Die abschließende Evaluation ergab, dass sich die hier vorgestellte Prüfungsverwaltung zur operativen Prozessunterstützung eignet. Allerdings wurde ebenfalls deutlich, dass durch weitere funktionale Erweiterungen und Aufklärung der Nutzer eine noch höhere Akzeptanz erreicht werden kann.

Abbildungsverzeichnis

Literaturverzeichnis

[ans90] *IEEE standard glossary of software engineering terminology.* 1990.

[BW06] Thorsten Berger and Heinz-Werner Wollersheim. *Eine dienste- und komponentenbasierte Architektur zur elektronischen Durchführung von Prüfungen und zum Management von Lehrveranstaltungen.* 2006.

[Den] Ernst Denert. *Software-Engineering.* Springer, Berlin 1991,1992.

[fBF] Bundesministerium für Bildung and Forschung. *Der Bologna-Prozess.* `http://www.bmbf.de/de/3336.php`.

[Fro09] Marvin Frommhold. *Integration des Aufgaben-Frameworks in die LMS-Plattform OLAT.* Abt. Betriebliche Informationssysteme, Institut für Informatik, Universität Leipzig, 2009.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[Grä09] Hans-Gert Gräbe. *Anforderungsanalyse - Projekt PA-Modul XMAN.* Betriebliche Informationssysteme, Institut f. Informatik, Universität Leipzig, 2009.

[GRSF] Florian Gnägi, Sandra Roth, Renata Sevcikova, and Joël Fisler. *OLAT 6 - Funktionsübersicht.* `http://www.olat.org/website/en/download/OLAT_6_0_Funktionsuebersicht.pdf`.

[RFH+] Sandra Roth, Joël Fisler, Sandra Hübner, Christian Meier, and Sven Morgner. *OLAT 6 - Benutzerhandbuch.* `http://www.olat.org/website/en/download/help/OLAT_6_1_Manual_DE_print_090306.pdf`.

[SJ06] M. Stock and F. Jost. Olat Web GUI Framework. Technical report, JLS goodsolutions GmbH, 2006.

[SM02] Inc. Sun Microsystems. *Core J2EE Patterns - Data Access Object.* 2002.

[Som07] Ian Sommerville. *Software Engineering.* Addison-Wesley, 2007.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter der Verwendung der angegebenen Hilfsmittel angefertigt habe.

Leipzig, 30.08.2009

Daniel Gerber