Softwaretechnik-Praktikum SS 2010

swp10-1
Projektleiter: Robert Heinecke | Dokument erstellt von Katrin Karlisch und Erik Schreiber

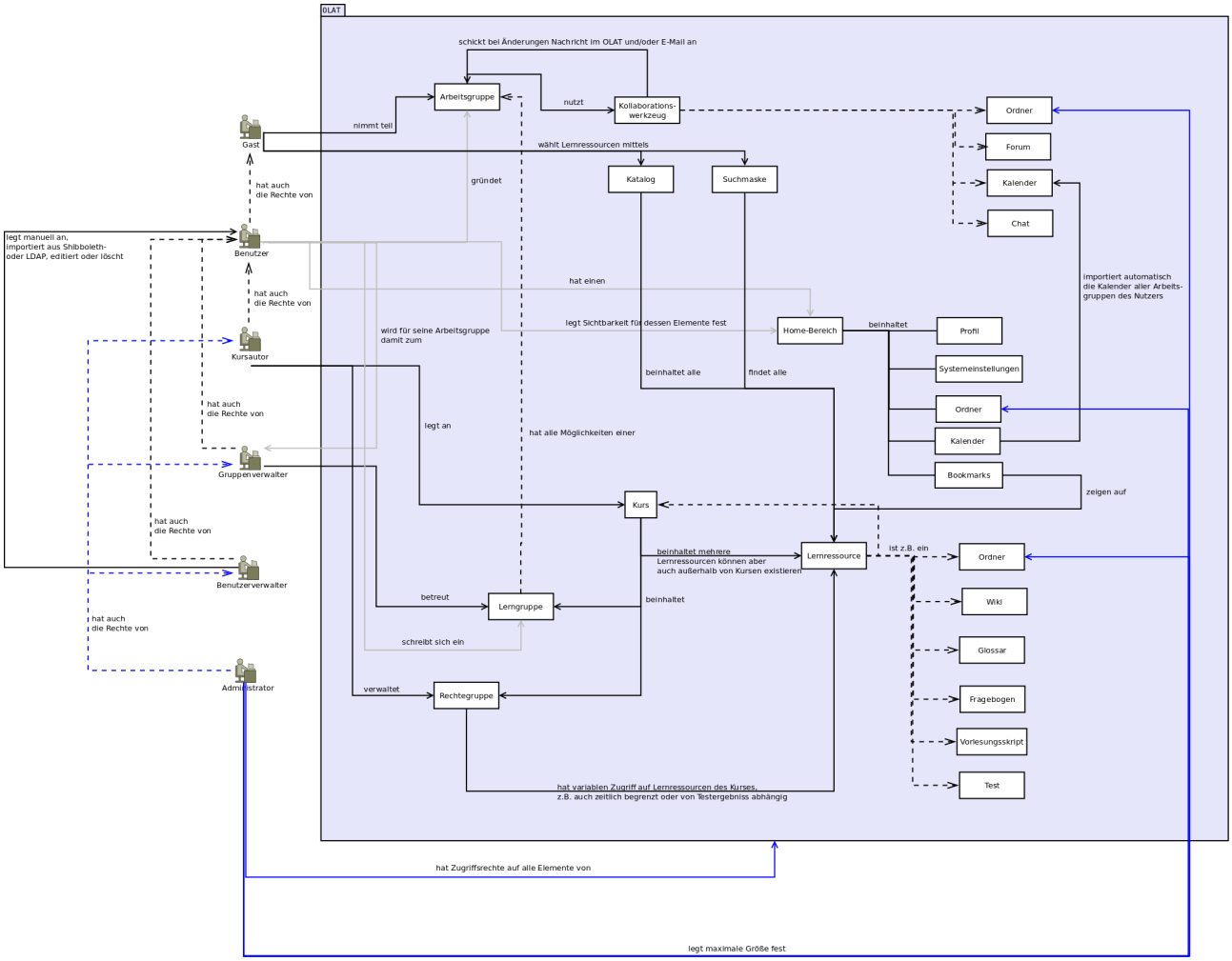
Entwurfsbeschreibung des Basisprojekts

1. Allgemeines

OLAT (**O**nline **L**earning **a**nd **T**raining) ist ein Learning Management System (LMS), welches zur Organisation von Lehrveranstaltungen genutzt wird. Das System ermöglicht das Bereitstellen von Lernressourcen zu den einzelnen Lehrveranstaltungen, eine Kontrolle des Lernfortschritts und eine Vereinfachung der Verwaltung, sowie des Lehr- und Lernbetriebs.

Das LMS OLAT ist in der Programmiersprache Java geschrieben, was einen Betrieb unter verschiedenen Betriebssystemen wie Windows, Mac OSX, Linux, BSD, Unix oder Solaris ermöglicht, ohne Anpassungen vornehmen zu müssen. Zur Speicherung der Daten können verschiedene Datenbankmanagementsysteme wie beispielsweise MySQL oder Oracle eingesetzt werden. OLAT verwendet eine Servlet-basierte Architektur. Darstellungslogik, Ablauflogik, Businesslogik und die Datenhaltung werden strikt getrennt. Dies geschieht durch ein eigens entwickeltes Model-View-Controller Framework.

2. Produktübersicht



OLAT ist ein E-Learning Portal, welches zahlreiche Funktionen bietet, um das Lernen und Organisieren des Studiums zu vereinfachen und übersichtlicher zu gestalten.

Lernressourcenverwaltung:

Das in OLAT integrierte Kurssystem ermöglicht Benutzern das Einschreiben und Austragen in Kurse, sowie in zugehörige Gruppen. Kursautoren können ihre Kurse, Benutzer, Gruppen, sowie deren Rechte verwalten. In jeden Kurs können zusätzliche Komponenten wie Wikis, Foren, Links oder Chats eingebunden werden. Ganze Kurse lassen sich via SCORM, IMS CP oder WTI importieren bzw. exportieren. Eingeschriebene Benutzer erhalten so Zugriff auf etwaige Lernressourcen. Beispiele für solche Lernressourcen wären z.B. Vorlesungsskripte, Übungsblätter, Tests oder aber auch themenverwandte Links oder Zusatzliteratur.

Kollaboration:

Um sämtliche Möglichkeiten auszunutzen, welche das Internet bietet, ist es im OLAT möglich externe Inhalte mit Kursen zu verknüpfen. Hierzu werden viele Elemente des modernen Internets auf den Lehrbetrieb der Universität abgebildet. Beispiele dafür sind unter Anderem:

a) Diskussionsforen
b) Wikis
c) Hausaufgabenverteilung und -abgabe
d) Tests und Selbsttests,
e) Fragebögen (Evaluierung)
f) Lerngruppen
g) Chat (via XMPP).

Administration:

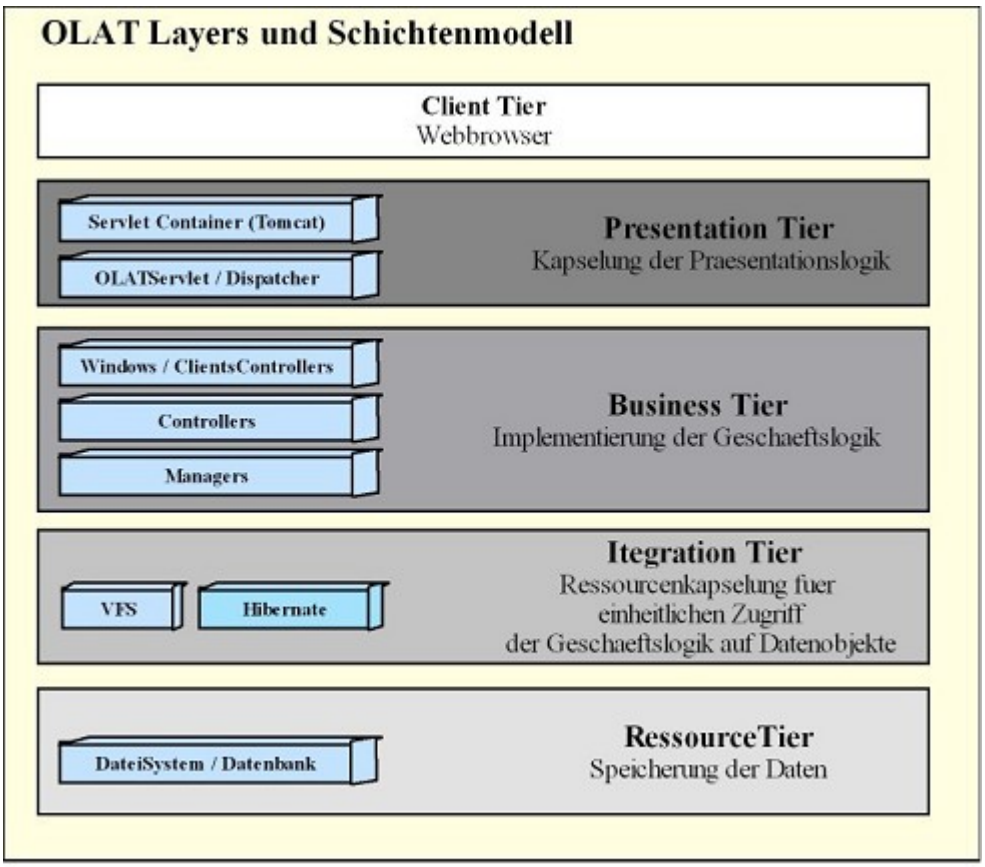
Die Plattform setzt auf die im Web üblichen Benutzerrollen Administrator, verschiedene Autorenarten und normale User. Kursleiter, Professoren, etc. entsprechen den Autoren und sind in der Lage ihre Kurse zu verwalten und neue Inhalte bereit zu stellen. Neben der umfassenden Kurs- und Benutzeradministration, ist auch ein Gastzugang prinzipiell möglich, kann aber dank umfassender Konfigurationsmöglichkeiten auch deaktiviert werden. Zudem werden mit OLAT bereits einige Analyse-Tools mitgeliefert, mit denen z.B. die Auslastung des Servers oder die Benutzeraktivität anschaulich gemacht werden können. Des weiteren kann mittels der Logfunktion alles geloggt und überprüft werden, was vor allem für Debugzwecke äußerst nützlich ist.

Persönliches:

Jeder Benutzer verfügt über einen persönlichen Bereich. Hier können diverse Einstellungen vorgenommen, Leistungsnachweise eingesehen oder eigene Dateien und Ordner verwaltet werden. Außerdem gibt es die Möglichkeit Termine und Veranstaltungen in einen persönlichen Kalender einzutragen, Notizen zu hinterlassen, Lesezeichen auf Kurse im OLAT zu verwalten oder mit anderen Benutzern zu in Kontakt zu treten.

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Umsetzung des Schichtenmodells in OLAT

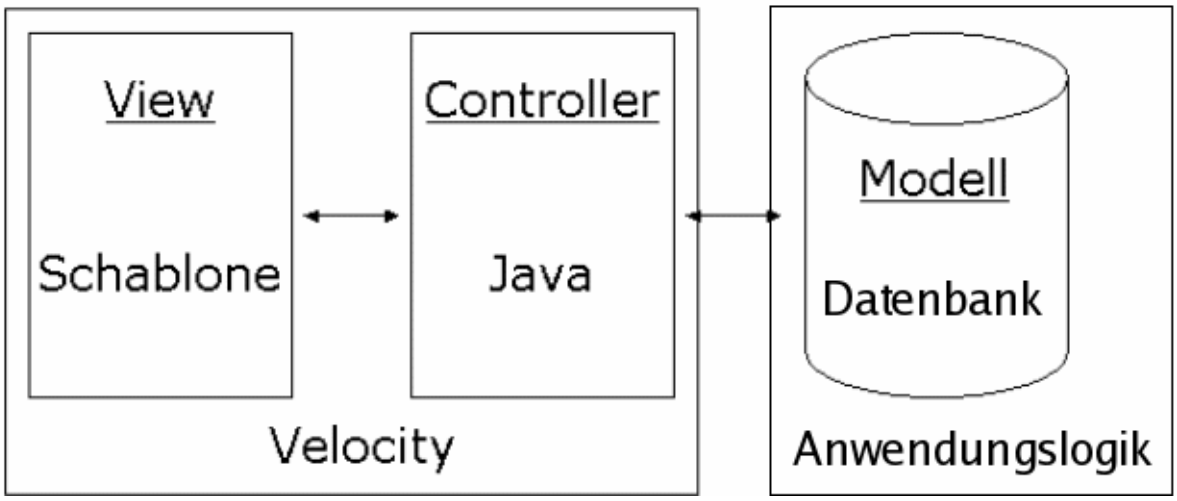


Innerhalb von OLAT wird das über J2EE spezifizierte Schichtenmodell umgesetzt. Jede Schicht ist in sich geschlossen und nur über definierte Schnittstellen ansprechbar. Dadurch sind die Schichten einzeln austauschbar und Abhängigkeiten zwischen ihnen werden minimiert, was zu einem entkoppelten System führt.

Den Sockel jeder Webapplikation nach J2EE bildet das „Resource Tier“. In diesem werden alle zu speichernden Daten abgelegt. Diese Ressourcen werden vom „Integration Tier“ gekapselt, welches auch dafür sorgt, dass die Geschäftslogik im „Business Tier“ über eine einheitliche Schnittstelle darauf zugreifen kann. Das „Presentation Tier“ ist dafür zuständig, die von der Geschäftslogik übermittelten Inhalte in ein Format zu überführen, das der Client interpretieren und darstellen kann.

Unter Berücksichtigung dieses Schichtenmodells lässt sich das OLATZ-Modell in sechs Ebenen gliedern. Die unterste Ebene bildet das Datenbankmanagementsystem. Auf dieses setzt das Hibernate-Persistenz-Framework auf, welches als Integrationsschicht zwischen Datenbank und Applikation fungiert. Eine zusätzliche Datenbank-Zugriffs-Ebene dient als Schnittstelle zwischen der eigentlichen Geschäftslogik im engeren Sinn und den persistenten Daten. Die Applikationslogik ist im Wesentlichen für die Umsetzung der Funktionalität verantwortlich. Die darauf aufbauende Ebene der Präsentationslogik bereitet mittels Velocity die Daten für den Client so auf, dass dieser innerhalb der Schicht der Präsentation die Daten lediglich darzustellen braucht.

Umsetzung des Model-View-Controller-Konzepts (MVC) in OLAT



Über das Velocity-Konzept erfolgt die Umsetzung des Model-View-Controller-Konzepts, das eine Trennung zwischen Präsentations- und Applikationslogik vorsieht. OLAT baut seine Webseiten nach dem Container-Baustein-Prinzip auf. Eine Seite besteht dabei aus einem Container, welcher Darstellungsbausteine und weitere Container in sich aufnehmen kann. Hauptsächlich verwendete Container sind das Panel (wird lediglich als Hülle um einen Inhalt verwendet) und der Velocity-Container. Jede OLAT-Webseite besteht auf oberster Ebene aus einem Main-Panel, welches grundsätzlich einen Velocity-Container enthält, der drei Komponenten beherbergt: ein Header-Panel, ein Content-Panel und ein Footer-Velocity-Container. Die Präsentationsebene (View) wird in Form einer Schablone (Template) realisiert und dient nur der Darstellung bzw. Gliederung einer Webseite. Die Geschäftslogik wird in Java-Klassen umgesetzt(Controller-Ebene). Um die Präsentationsschicht von der Geschäftslogik zu trennen, nutzt OLAT nun das Velocity-Konzept. Dabei werden HTML-Anweisungen aus einer Schablone (Template) gelesen, in welcher Struktur und Aufbau der Webseite festgelegt sind. Dadurch hat die Geschäftslogik keine Einflüsse auf das Layout und widmet sich nur der Umsetzung der Funktionalität.
OLAT nutzt zur Umsetzung des Velocity-Konzepts mehrere Java-Klassen, in denen die Funktionalitäten gekapselt werden:
Im Konstruktor der Klasse org.olat.gui.render.velocity.VelocityHelper erfolgt die Initialisierung der Velocity-Engine. Dies geschieht bei Ausführung der init()-Methode des OLAT-Servlets, von dem aus alle weiteren Klassen zur Bearbeitung der Anfragen eines Clients angestoßen werden. Die Klasse org.olat.gui.components.VelocityContainer nimmt genau ein Template in sich auf und erzeugt und verwaltet das zugehörige Kontextobjekt.
Welches Template benutzt wird und welche Schlüssel in das Kontextobjekt aufgenommen werden, entscheidet die Applikationslogik (Controller-Klassen). Das Zusammenführen von Template und zugrunde liegendem Kontextobjekt erfolgt über die Klasse VelocityHelper, welche ebenfalls die Velocity-Engine verwaltet.

Projektleiter: Robert Heinecke | Dokument erstellt von Katrin Karlisch und Erik Schreiber

Umsetzung des Business Delegate Prinzips

Business Delegate-Prinzip allgemein

Business Delegate ist ein Java-EE-Entwurfsmuster. Das Business Delegate Pattern wird verwendet, um die Präsentationsschicht (Presentation Tier) von der Geschäftslogik (Business Tier) zu entkoppeln. Hierzu wird eine Schnittstelle bereitgestellt, deren Methoden die Geschäftslogik der Anwendung definieren, und damit anzeigen, was die Anwendung eigentlich tut.

Der Business Delegate verwendet dann intern die Komponenten der Geschäftslogik und leitet Aufrufe der Präsentationslogik an sie weiter. Werden Änderungen in der Implementierung der Business-Schicht vorgenommen, müssen nun nicht mehr alle Elemente der Präsentationsschicht geändert werden, sondern nur noch die Business-Delegate-Klassen.

zur Umsetzung in OLAT :

In der Applikationslogik werden Nutzeraktionen verarbeitet und das Main-Panel mit neuem Inhalt gefüllt. Allen Komponenten (Darstellungsbausteine, Container) des Main-Panels liegen Controller-Klassen zugrunde, die auf Events (Nutzeraktionen) „lauschen“. Nur den Panels selbst liegen keine Controller zu Grunde. Wenn eine Nutzeraktion stattgefunden hat, erfolgt die Identifizierung der entsprechenden Controller über die Komponenten, in denen das Event stattgefunden hat. Die Controller (auch Listener) generieren dann entsprechend der Nutzeraktion eine “Antwort”. Alle an den Server bzw. das System gestellten Anfragen werden vom Servlet org.olat.servlets.OLATServlet entgegengenommen.

Welcher Dispatcher sich nun der Anfrage zu widmen hat, spezifiziert der Teil der URL nach /olat/. Je nach ausgelöstem Event werden weitere Controller angestoßen, die das Main-Panel oder Teile davon mit neuem Inhalt füllen, d.h. neue Container und Darstellungsbausteine anlegen und in das Main-Panel einbauen.

Welcher Controller gerade auf welche Komponente hört, kann durch den Debugmodus auf Präsentationsebene herausgefunden werden. Zusammenfassend zur Applikationslogik: Der Controller der auf eine Komponente „lauscht“, legt diese in seinem Konstruktor auch selbst an. Wenn ein Event innerhalb einer Komponente ausgelöst wird, erfolgt ein Aufruf der event()-Methode des auf diese Komponente „lauschenden“ Controllers und das Ereignis wird dort verarbeitet.

Kapselung der DB -Zugriffslogik:

OLAT verwendet für den Zugriff auf die Datenbank das Hibernate Framework. Dieses ermöglicht es, Objekte in einer relationalen Datenbank zu speichern und umgekehrt aus Datensätzen einer relationalen Datenbank Objekte zu erzeugen. Die Datenbank ist dabei beliebig wählbar und austauschbar, sofern ein JDBC-Treiber dafür existiert.

Hibernate steht als Mittlerschicht zwischen der eigentlichen Applikation und der Datenbank. Die Anwendung des Hibernate Frameworks wird über Manager-Klassen gekapselt.

4. Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

Zugriff auf die Datenschicht

Für Daten, die in der Applikationslogik entstehen bzw. vom Nutzer eingegeben wurden, ist es notwendig, diese persistent in einer Datenbank abzulegen, um bei Bedarf wieder darauf zurückzugreifen. Nebst den Daten auf der Datenbank werden gewisse Dateien aus Performance- und Strukturgründen auf dem Filesystem gehalten. Der Primärschlüssel eines Datenbank Eintrags wird als Name des Ordners verwendet, um die Eindeutigkeit zu garantieren.
OLAT verwendet für die Arbeit mit der Datenbank das für Java entwickelte Open-Source-Persistenz-Framework Hibernate. Innerhalb dieses Frameworks können Objekte der Applikationsebene (so genannte persistente Objekte) auf Datenbankrelationen abgebildet werden. Dabei entsprechen die Membervariablen eines persistenten Objektes den Attributen einer Relation.

Wie bereitet OLAT den Zugriff auf die Datenbank und das Zusammenspiel mit dem Hibernate-Framework vor, welche persistenten Klassen stehen zur Verfügung und wer verwaltet diese?

Persistente Klassen sind nach einem einheitlichen Prinzip strukturiert. Sie besitzen Membervariablen, welche die eigentlichen Datenfelder bilden und im Konstruktor gegebenenfalls initialisiert werden. Für jede Variable gibt es eine Getter- und eine Setter-Methode, die diese ausliest bzw. setzt.

Die in OLAT verwendeten persistenten Klassen implementieren das Interface org.olat.core.commons.persistence.PersistentObject. Dieses sorgt dafür, dass alle persistenten Klassen einen eindeutigen Schlüssel und die zwei Timestamps „creationDate“ sowie „lastModified“ erhalten. Der Schlüssel entspricht dem Primärschlüssel der Relation, auf welche das Objekt abgebildet wird. Dadurch sind alle Datensätze innerhalb der Relation eindeutig identifizierbar.
Über Manager-Klassen werden persistente Objekte verwaltet, in der Regel operiert genau ein Manager mit einer persistenten Klasse.

Einige Beispiele sind:
• org.olat.user.UserManager für UserImpl
• org.olat.properties.PropertyManager für Property
• org.olat.catalog.CatalogManager für CatalogEntryImpl
• org.olat.repository.RepositoryManager für RepositoryEntry

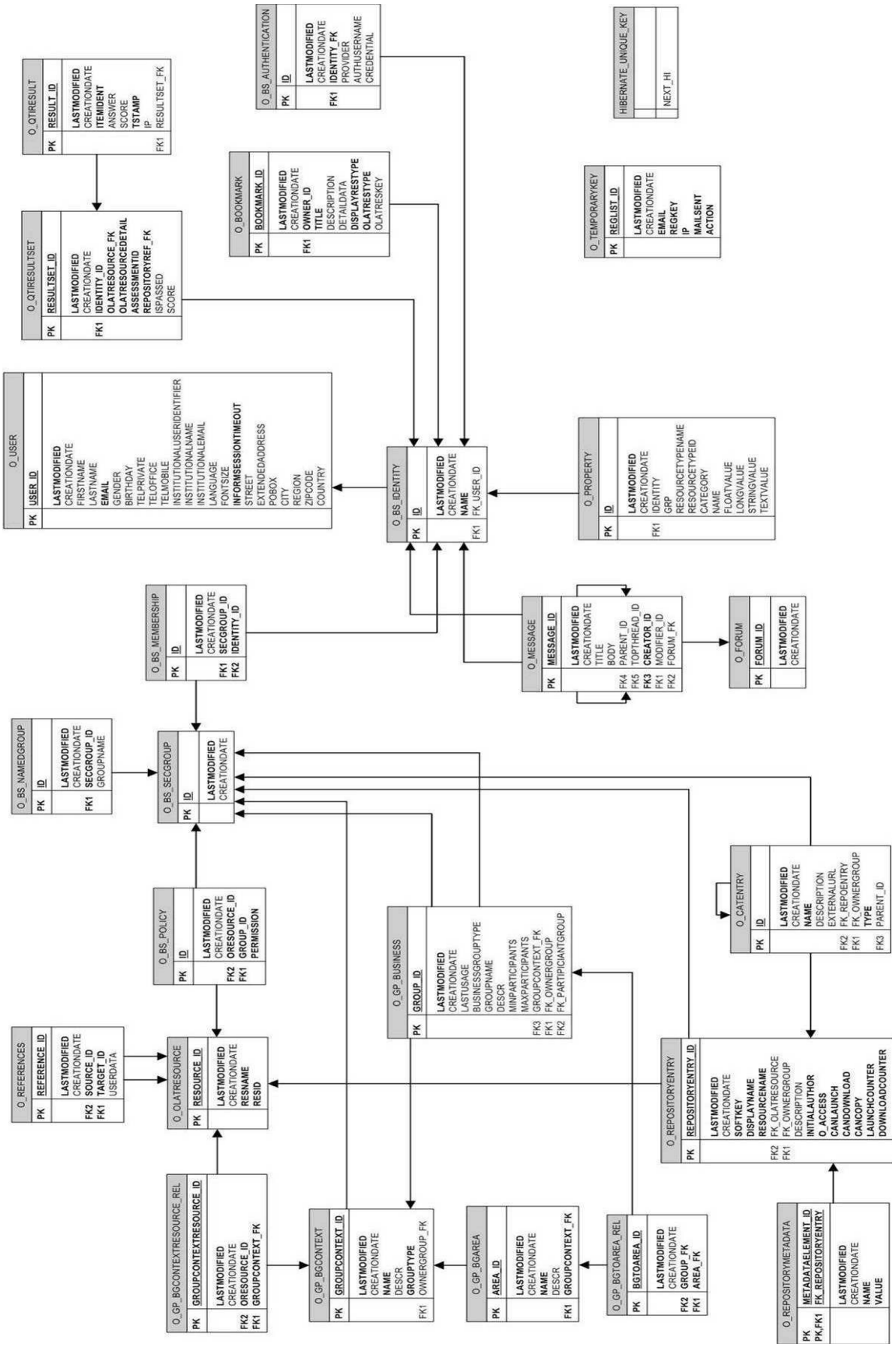
Die Controller aus der Applikationsschicht oder in seltenen Fällen Darstellungsbausteine oder Container nehmen die Funktionalität der Manager in Anspruch. Manager legen neue persistente Objekte an, füllen sie mit Werten aus der Applikationslogik und schreiben sie dann in die Datenbank. Umgedreht lesen sie Werte aus der Datenbank aus und stellen sie in Form von persistenten Objekten der Applikationslogik zur Verfügung. Die spezifischen Manager-Klassen nutzen nicht das OLAT-Framework, sondern greifen dafür auf org.olat.core.commons.persistence.DBManager zurück. Der DBManager nutzt die vom Hibernate-Framework zur Verfügung gestellten Hilfsmittel, um mit der Datenbank kommunizieren zu können. Einen Überblick hierüber gibt das Klassendiagramm Anhang 1.

Datenbankmanagementsystem

OLAT arbeitet mit dem Open-Source-Datenbankmanagementsystem MySQL. Neben den Vorzügen dieser Datenbank hat diese aber auch Nachteile. So ist es beispielsweise nicht möglich, Fremdschlüssel zu definieren, die eine referenzielle Integrität gewährleisten würden. Allerdings bietet MySQL die Möglichkeit, externe Tabellenformate wie zum Beispiel InnoDB zu verwenden, die dieses Problem lösen.Davon macht auch OLAT Gebrauch und bietet dadurch gleichzeitig eine Transaktionsunterstützung. im OLAT-Datenbankschema werden die Beziehungen unter den einzelnen Relationen deutlich:

Softwaretechnik-Praktikum SS 2010

swp10-1
Projektleiter: Robert Heinecke | Dokument erstellt von Katrin Karlisch und Erik Schreiber



Erweiterungspunkt für den Zugriff auf die Datenschicht

Die Datenbank wird als einziger Zugriffspunkt für Datenbankoperationen benutzt. Der Zugriff auf diese wird mittels dem Hibernate-Framework umgesetzt.
Das Hibernate-Framework ermöglicht es, Objekte in einer relationalen Datenbank zu speichern und umgekehrt aus Datensätzen einer relationalen Datenbank Objekte zu erzeugen. Die Datenbank ist dabei beliebig wählbar und austauschbar, sofern ein JDBC-Treiber dafür existiert. Hibernate dient somit der Abstraktion der Datenbasis und steht als Mittlerschicht zwischen der eigentlichen Applikation und der Datenbank.

Um seine Funktionsweise vollständig entfalten zu können, benötigt das Framework folgende Komponenten:

a) eine Hibernate-Konfiguartionsdatei (hibernate.properties bzw. hibernate.cfg.xml),

b) die Hibernate Java-Library,

c) die Hibernate Query Language (HQL) für alle Datenbankabfragen,

d) die zu speichernden Java-Objekte (persistente Klassen),

e) XML-Mapping-Dateien für die persistenten Klassen sowie

f) einer Datenbank samt Datenbankschema.

Die Konfiguration des Hibernate-Persistenz-Frameworks erfolgt textbasiert über die Datei hibernate.properties. Alternativ dazu kann dies auch mittels XML über hibernate.cfg.xml gelöst werden. Die XML-Mapping-Dateien bilden die persistenten Klassen auf Datenbankrelationen ab. Im besten Falle wird jede persistente Klasse durch genau eine Datenbankrelation repräsentiert, wobei die Membervariablen der Klasse die Attribute der Relation bilden. OLAT hinterlegt die Mapping-Dateien stets im gleichen Verzeichnis wie die persistente Klasse und benennt jeweils beide Dateien gleich. Eine Zuordnung von persistenter Klasse zur Mapping-Datei ist somit schneller möglich.

Konkreter Erweiterungspunkt ist die Klasse org.olat.persistence.DB (in unserem Fall: org.olat.core.commons.persistence), mit welcher neue Hibernate-Mappings hinzugefügt werden können.

Erweiterung von OLAT um eine externe Datenquelle (z.B. OD) anzubinden

Ein Erweiterungspunkt dieser Art wäre für unser Projekt zwar äußerst hilfreich, existiert aber noch nicht. Die Ausgestaltung wäre zudem relativ komplex. Das Problem stellt die interne Repräsentation der Daten im OLAT dar:
Die OD ist eine Datenquelle im RDF Format. Um einzelne “Datensätze” anzusprechen ist daher im allgemeinen ein Traversieren des RDF-Graphen notwendig. Dies ist für die Nutzung im OLAT allerdings wenig praktikabel, da für jede Operation eine Anfrage an die externe Datenquelle notwendig wäre. Damit verbunden wäre eine schlechte Skalierbarkeit, da der Server der externen Datenquelle zum „Flaschenhals“ werden könnte. Gewünscht ist hier eine „Importfunktion“, um die externen Daten in die interne, relationale Datenbank zu übernehmen und deren oben beschriebene Vorteile nutzen zu können.

Hieraus ergeben sich zwei Probleme:

- In der internen Datenbank müssen neue Tabellen erstellt werden, um Hibernate die Persistierung der Daten zu ermöglichen. Dies ist durch die Hibernate Umgebung aber nicht trivial. Zwar könnten auch mit Hibernate über SQL Statements - im jeweiligen Dialekt der Datenbank – direkt im Code Tabellen erstellt werden, dies bedeutet aber eine schlechte Übersichtlichkeit und widerspricht zudem der Trennung im Schichtenmodell. Die Lösung der Wahl ist zur Zeit ein

neuer Eintrag in OLATs setupDatabase.sql im jeweiligen Datenbank Dialekt, was wiederrum dem Extension-Konzept widerspricht. Folglich ist der erste Anspruch an den Erweiterungspunkt die Erstellung neuer Tabellen mit automatischem Eintrag in die setupDatabase.sql. Entsprechende Hibernate Mappings können dann über org.olat.extensions.hibernate.HibernateConfigurator erstellt werden.

- Die Struktur dieser Tabellen ist von der Struktur des RDF-Graphen abhängig. Unseres Wissens nach gibt es derzeit noch kein automatischen Verfahren zur Konversion von RDF-Graphen zu relationalen Datenbanken. Weiterhin wird also eine Schnittstelle benötigt, die SPARQL Anfragen an die externe Datenquelle stellt und die Ergebnisse der Anfrage dann als Java-Objekte über Hibernate persistiert. Dazu werden oben erwähnte Tabellen benötigt.

Der Erweiterungspunkt müsste also in der Lage sein, für Datenobjekte aus der externen Datenquelle neue Tabellen in der OLAT Datenbank zu erstellen, (und eventuell entsprechende Hibernate Mappings) und zusätzlich eine Schnittstelle anbieten, an die eine Anfrageklasse in der Sprache der externen Datenquelle andocken kann. Diese Klasse würde dann den eigentlichen “Adapter” zwischen dem Format der externen Datenquelle und der internen relationalen Datenbank darstellen und durch Abfragen auf der externen Datenbank Java Objekte erzeugen, deren Struktur sich in den jeweiligen Tabellen und Mappings widerspiegeln würde. Expizit für die OD und die Struktur ihrer von uns benötigten Datenobjekte tun wir prinzipiell genau das, ein Extension Point beschriebener Art und Ausprägung wäre aber in Hinblick auf die weitere Entwicklung des Web und eventuell neu hinzukommende Datenbankformate durchaus sinnvoll.

Anhang 1: Klassendiagramm DB Interaktion