Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1

Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

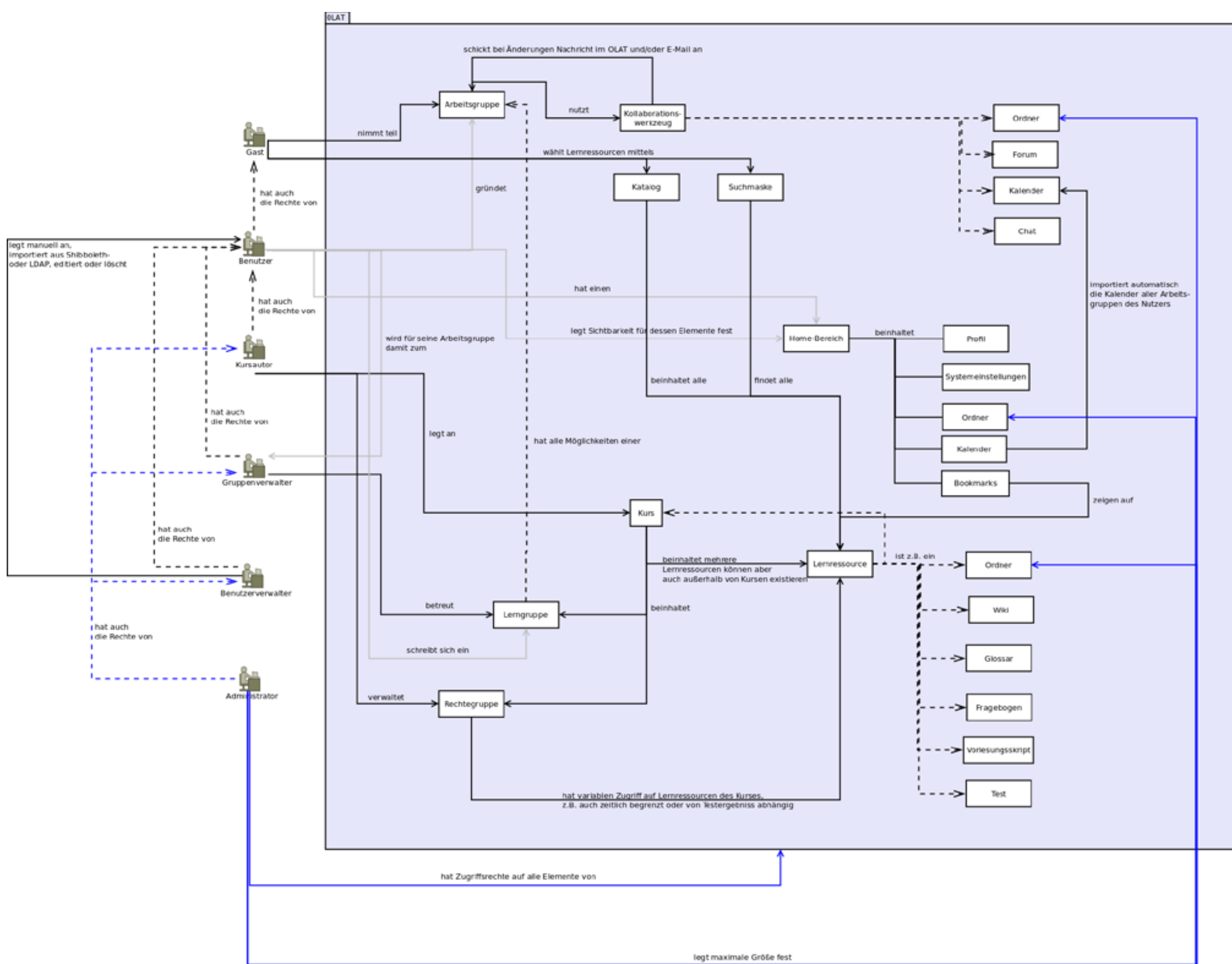
Entwurfsbeschreibung des Prototypen der Gruppe swp11-1

1. Allgemeines

Das webbasierte Learning Management System OLAT unterstützt das kollaborative Arbeiten in Gruppen und bietet ein flexibles Kurssystem, welches durch eine mehrsprachige Benutzeroberfläche und verschiedene Groupwaretools realisiert wird.

In unserem Projekt wird nun Olat um ein weiteres Modul erweitert. Das Prüfungsmodul soll Studenten und Prüfern das Erstellen und Verwalten von Prüfungen erleichtern. Es wird Studenten die Möglichkeiten bieten sich zu einer Prüfung an-/abzumelden und Prüfern ermöglichen diese zu erzeugen, zu verwalten und bekannt zu geben.

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.

Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

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.

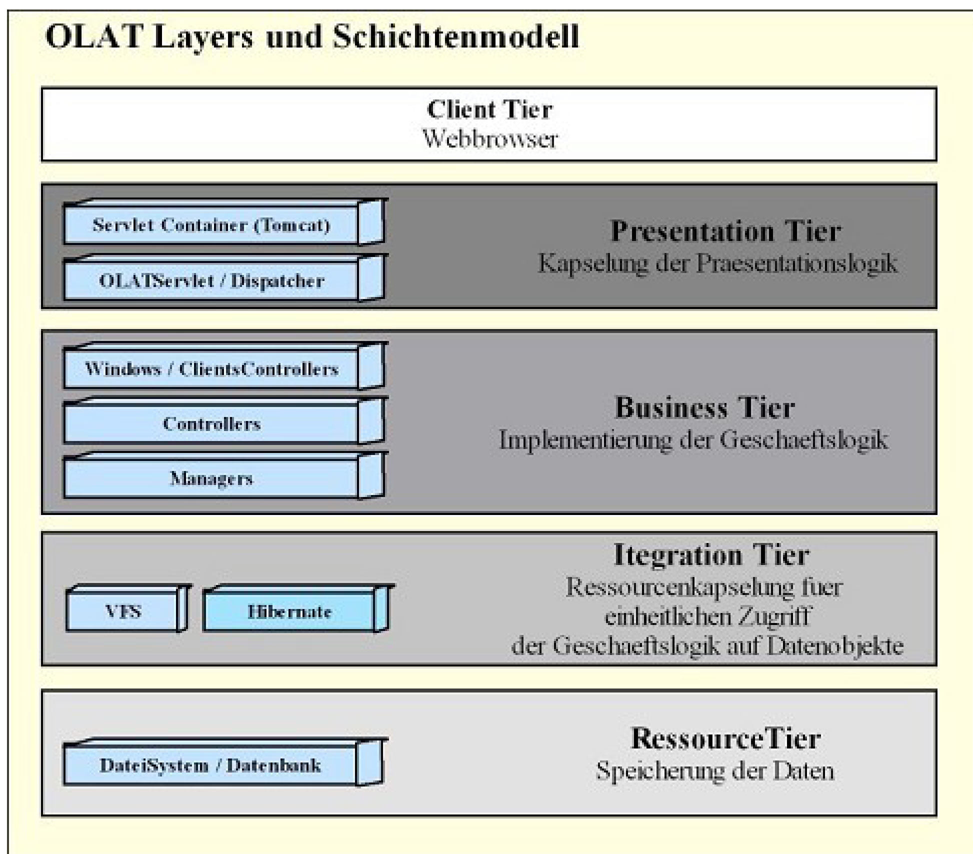
Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

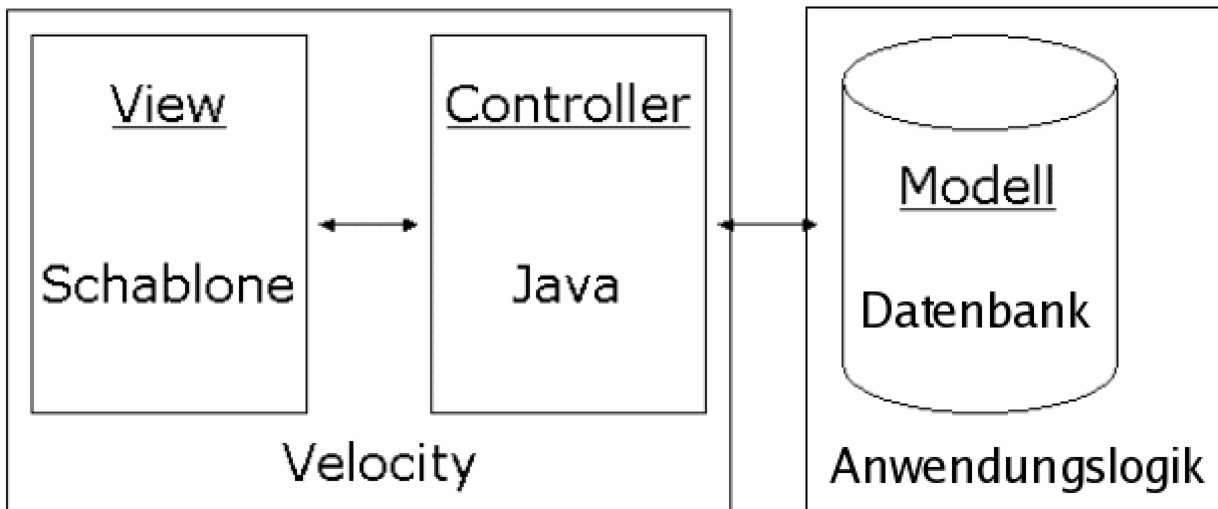
3.1 Umsetzung des Schichtenmodells in OLAT



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 OLAT-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.

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

3.2 Umsetzung des MVC-Konzepts 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.

3.3 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.

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.

3.4 Core Concerns und Cross Cutting Concerns

Core Concerns(„Funktionalität für die ein Programm geschrieben wurde"):

Core Container(Core, Beans, Context und Expression Language Module)

Core und Beans:
- stellen die grundlegenden Teile des Frameworks bereit (darin eingeschlossen IoC (Inversion of Control) und DI(Dependency Injection))
- *BeanFactory* ist verfeinerte implementation des *factory pattern* (erlaubt Entkopplung von Konfiguration und Spezifikation von Abhängigkeiten der aktuellen Programmlogik)

Context Modul:
- baut auf von Core und Bean bereitgestellten Basis auf
- ist ein Mittel um Objekten Zugang zu Framework im Stil von JNDI registry zu verschaffen (JNDI *Java Naming and Directory Interface*)
- erbt Features von *Beans Modul* und fügt Unterstützung für Internationalisierung, Event-Bekanntmachungen, Resourcen Laden und transparentes Erstellen von Context (Z.B.: Servlet Container) hinzu
- unterstützt auch Java EE features wie EJB(Enterprise JavaBeans), JMX(Java Management Extensions) und Basic Remoting
- *das ApplicationContext* interface ist Zentrum des Context Moduls

Expression Language:
- bietet mächtige „expression language" für querying und Ändern eines Objectgraphen zur Laufzeit

- ist Extension der *unified expression language* (unified EL)
- unterstützt Setzen und Beziehen von Eigenschaften und Inhalten,
- Eigenschaftszuweisungen, Methodenaufruf, Zugriff auf den Kontext von Arrays, Collections und Indexe, logische und arithmetische Operatoren, Variablennamen, und Abrufen von Objekten nach Namen von Spring's IoC container
- unterstützt ebenso Listen-Projection und Selection sowie common list aggregations

Data Access/Integration (JDBC, ORM, OXM, JMS und Transaction Module):

- JDBC Modul stellt JDBC-abstraktions-Schicht zur Verfügung die die Arbeit des coding und parsing Datenbank-Vendor-spezifischen Fehlercodes abnimmt
- ORM Modul stellt Integrations-Schichten für Objekt-Relationales Mapping, einschließlich JPA, JDO, Hibernate, und iBatis bereit (durch ORM package sind alle diese O/R-Mapping Frameworks nutzbar)
- OXM Modul stellt Abstraktionsschicht bereit, welche Object/XML Mapping Implementierungen für JAXB, Castor, XMLBeans, JiBX und Xstream unterstützt
- JMS (Java Messaging Service) Modul beinhaltet features zum Erstellen und Empfangen von Nachrichten
- Transaction Modul unterstützt programmatisches und deklaratives Transaktions Management für Klassen die spezielle Interfaces implementieren und für alle „POJOs“ (plain old Java objects)

Web (Web, Web-Servlet, Web-Struts, und Web-Portlet Module):

- Web Modul stellt grundlegende Web-orientierte Integrations-Features wie „multipart file-upload functionality“ und die Initialisierung des IoC Containers unter Verwendung von „servlet listeners“ und eines web-orientierten application context s bereit(beinhaltet ebenso Web-bezogene Teile des Spring Remote Supports)
- Web-Servlet Modul beinhaltet Spring's Model-View-Controller Implementation für Web Applications (stellt Unterscheidung zwischen Domain Model Code und Web Forms bereit, und integriert mit allen anderen Features des Spring Framework)
- Web-Struts Modul beinhaltet Klassen zur Unterstützung des Integrierens einer klassischen „Struts web tier“ in eine Spring Application
- Web-Portlet Modul stellt MVC Implementierung bereit um in einer Portlet-Umgebung genutzt zu werden und spiegelt die Funktionalität des Web-Servlet Modul wieder

AOP and Instrumentation :

AOP Modul stellt AOP Allianz-konforme Aspekt-orientierte Programmierungsumsetzung zur Verfügung die es z.B. erlaubt Methoden-Interceptor und Pointcuts zu definieren um solchen Code sauber zu entkoppeln, der Funktionalitäten implementiert die separiert sein sollten (siehe auch Cross Cutting Concerns)

Test:

Test Modul unterstützt das Testen von Spring Komponenten mit JUnit oder TestNG (stellt konsistentes Laden von Spring ApplicationContexts und caching dieses Contexts und mock Objekte, welche zum Testen von Code in Isolation genutzt werden, bereit)

Cross Cutting Concerns/CCC’s(„Methoden der Concerns schneiden sich“):

CCC’s werden in Spring durch das AOP (Aspect Oriented Programming) Framework realisiert (Modularisieren der CCC's durch Aspekte (aspects))

Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

AOP-Konzepte in Spring:

Pointcut API in Spring:
- *org.springframework.aop.Pointcut* interface ist zentrales interface, um advices zu bestimmten Klassen und Methoden zu bestimmen
- Spalten des Pointcut Interfafce in zwei Teile erlaubt wiederverwendung von übereinstimmenden Teilen von Klassen und Methoden und zusammengesetzten Operationen (z.B.: Vereinen mit anderen „method matchern")
- *ClassFilter* interface verwendet um Pointcut auf gegebene Menge von Zielklassen einzuschränken (wenn *matches()* Methode immer true zurückgibt, werden alle Klassen „gematched")
- *MethodMatcher* Interface : *matches(Method, Class)* Method zum Testen ob der Pointcut jemals eine gegebene Methode zur Zielklasse „matcht"
- Operationen: *union* (einer der Pointcut matcht) und *intersection* (beide Pointcuts matchen)

Advice API in Spring :
- Jedes Advice ist Spring bean
- Advice Instanz kann mit allen „advised objects" geteilt werden, oder einzigartig sein
- entspricht *per-class* oder *per-instance* advice
- Typen: Interception around advice, Before advice, Throws advice, After Returning advice, Introduction advice

Advisor API in Spring :
- Aspekt mit nur einem Advice Objekt, dass mit einem Pointcut verbunden ist
- *org.springframework.aop.support.DefaultPointcutAdvisor* ist am haufigsten verwendete Advisor Klasse

Verwenden des ProxyFactoryBean um AOP Proxies zu erstellen :
- klassischer Weg um AOP proxy in Spring zu ertellen ist *org.springframework.aop.framework.ProxyFactoryBean zu verwenden*

Präzise Proxy Definitionen :
- „Eltern", *template*, Bean Definition wird für den Proxy erzeugt (aber nie selbst instantiiert)
- jeder Proxy der erstellt werden muss ist „child bean" definiert

Programmarisches Erstellen von AOP Proxies mit der ProxyFactory :
- Erstellen eines Objektes vom Typ *org.springframework.aop.framework.ProxyFactory*

Manipulieren von „advised objects" :
- Ändern von proxies unter Verwendung vom *org.springframework.aop.framework.Advised* Interface

Verwenden der „autoproxy facility" :
- Einrichten einiger spezieller Bean Definitionen in XML Bean Definitions-Datei um auto proxy Infrastruktur zu konfigurieren
- *ProxyFactoryBean nicht benötigt*

Verwenden von TargetSources :
- Konzept des *TargetSource*, beschrieben durch *org.springframework.aop.TargetSource* Interface
- *TargetSource* bekommt Anfrage nach Zielinstanz, immer nach Berarbeitung eines Methodenaufrufs durch AOP Proxy

- Typen: Hot swappable target sources , Pooling target sources , Prototype target sources , ThreadLocal target sources

Definieren eines neuen Advice types :
- *org.springframework.aop.framework.adapter* Package ist SPI (Serial Peripheral Interface)Package (erlaubt hinzufügen von neuen praktischen „Advice Typen“ ohne Änderung des Core Framework) diese „Advice Typen“ müssen das *org.aopalliance.aop.Advice* tag Interface implementieren

3.5 Erweiterungspunkte

Erweiterungspunkt: org.olat.dispatcher.DispatcherAction
- Interface: org.olat.extensions.globalmapper
- Klassen: org.olat.dispatcher.DispatcherAction
- Beschreibung: Die Erweiterung kann als Mapper bezeichnet werden und kennt den Pfad mit welchem er assoziiert ist

Erweiterungspunkt: org.olat.home.HomeMainController
- Interface: org.olat.extensions.action.ActionExtension
- Beschreibung: Die Erweiterung liefert einen Link mit Beschreibung und Aktionsdefinition

Erweiterungspunkt: org.olat.gui.components.Window
- Interface: org.olat.extensions.css.CSSIncluder
- Klassen: org.olat.gui.css.CSSGenerator
- Beschreibung: Neue CSS-Stylesheets

Erweiterungspunkt: org.olat.persistence.DB
- Interface: org.olat.extensions.hibernate.HibernateConfigurator
- Klassen: org.olat.persistence.DB
- Beschreibung: Erweitert kann neue hibernate-mappings enthalten

Erweiterungspunkt: org.olat.gui.control.generic.dtabs.DTabs
- Interface: org.olat.extensions.sitescreator.SitesCreator
- Klassen: org.olat.FullChiefController
- Beschreibung: Neue Seiten; muss SitesCreator enthalten, der Liste von SiteDefinition Objekten enthält

Erweiterungspunkt: org.olat.login.AfterLoginInterceptorController
- Interface: org.olat.login.SupportsAfterLoginInterceptor
- Klassen: org.olat.user.UserModule
- Beschreibung: Liefert eine Möglichkeit direkt nach den Login in den Workflow einzugreifen

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

4. Grundsätzliche Struktur- und Entwurfsprinzipien für einzelne Pakete

4.1 Datenbankmanagementsystem

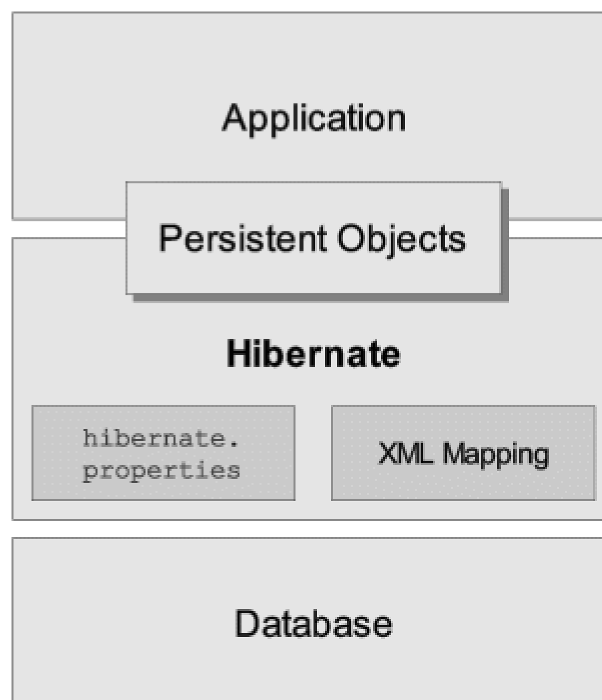
Die OLAT-Datenbank basiert auf dem Datenbankmanagementsystem MySQL. MySQL bietet die Möglichkeit externe Tabellenformate wie InnoDB zu verwenden, wovon OLAT auch gebraucht macht. Dadurch wird die referenzielle Integrität und Transaktionsunterstützung gewährleistet. Das OLAT-Datenbankschema verdeutlicht die Beziehungen unter den einzelnen Relationen (Anhang 1).

4.2 DB-Zugriffslogik in OLAT

OLAT speichert alle Daten die vom Nutzer eingegeben werden oder durch die Applikationslogik entstehen persistent in der zugrunde liegenden Datenbank. Zur Arbeit mit der Datenbank verwendet OLAT das so genannte Hibernate-Framework. Mit Hilfe dieses Frameworks können persistente Objekte (Objekte der Applikationsebene) auf Datenbankrelationen abgebildet werden. Dabei entsprechen die Membervariablen eines persistenten Objektes den Attributen einer Relation.

4.3 Das Hibernate-Framework

Mit Hilfe des Hibernate-Framework ist es möglich Objekte in einer relationalen Datenbank zu speichern und 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 Übergangsschicht zwischen der eigentlichen Applikation und der Datenbank. Die folgende Abbildung verdeutlicht die Eingliederung von Hibernate und zeigt die Grundbausteine des Frameworks:



Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther

Das Framework benötigt folgende Komponenten um die volle Funktionalität zu gewährleisten:

- eine Hibernate-Konfiguartionsdatei
- die Hibernate Java-Library
- die Hibernate Query Language (HQL) für alle Datenbankabfragen,
- die zu speichernden Java-Objekte (persistente Klassen)
- XML-Mapping-Dateien für die persistenten Klassen sowie
- 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. 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.
Datenbankabfragen werden über die Hibernate-eigene Sprache HQL realisiert. Diese ist in ihrer Syntax sehr ähnlich zu SQL, im Unterschied dazu aber folgt HQL dem objektorientierten Paradigma und beherrscht Vererbung, Polymorphie und Assoziation. Für die Realisierung der Datenbankzugriffe nutzt Hibernate eigene Sessions. Diese werde von der Klasse SessionFactory vergeben und verwaltet.

4.4 Wie greift OLAT mit Hilfe von Hibernate auf die Datenbank zu?

Die von OLAT verwendeten 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 get()- und eine set()-Methode
- Die in OLAT verwendeten persistenten Klassen müssen von der abstrakten Klasse PersistentObject abgeleitet werden (org.olat.core.commons.persistence), dadurch ist sichergestellt, dass alle persistenten Klassen einen eindeutigen Schlüssel, ein Erstellungsdatum und eine Versionsnummer erhalten. Der Schlüssel entspricht dem Primärschlüssel der Relation, auf welche das Objekt abgebildet wird. Dadurch sind alle Datensätze innerhalb der Relation eindeutig identifizierbar.

Die Verwaltung von persistenten Objekten erfolgt über so genannte Manager-Klassen, beispielsweise org.olat.user.UserManager oder org.olat.properties.PropertyManager.

Manager haben zwei wesentliche Aufgaben:

1. Anlegen neuer persistenter Objekte, die mit Werten aus der Applikationslogik gefüllt und anschließend in der Datenbank gespeichert werden.
2. Auslesen von persistenten Objekten aus der Datenbank, die sie der Applikationslogik zur Verfügung stellen.

Die spezifischen Manager-Klassen nutzen selbst nicht das OLAT-Framework, sondern greifen dafür auf org.olat.persistence.DBManager zurück, aus dem einfachen Grund, die immer wiederkehrenden Algorithmen nicht redundant implementieren zu müssen. Der DBManager nutzt die vom Hibernate-Framework zur Verfügung gestellten Hilfsmittel, um mit der Datenbank kommunizieren zu können. Das Klassendiagramm im Anhang zeigt die Wirkungsweise dieser Klassen.

4.5 org.olat.persistence.DB im Detail:

```
<<Interface>>
DB
----------------------------------------------------------------
+forceSetDebugLogLevel(enabled:boolean): void
+addTransactionListener(listener:ITransactionListener): void
+removeTransactionListener(listener:ITransactionListener): vo
+closeSession(): void
+cleanUpSession(): void
+createQuery(query:String): DBQuery
+deleteObject(object:Object): void
+find(query:String,value:Object,type:Type): List
+find(query:String,values:Object[],
      types:Type[]): List
+findObject(theClass:Class,key:Long): Object
+find(query:String): List
+saveObject(object:Object): void
+updateObject(object:Object): void
+delete(query:String,value:Object,type:Type): int
+delete(query:String,values:Object[],
        types:Type[]): int
+loadObject(persistable:Persistable): Persistable
+loadObject(persistable:Persistable,
            forceReloadFromDB:boolean): Persistable
+commitAndCloseSession(): void
+commit(): void
+rollbackAndCloseSession(): void
+rollback(): void
+getStatistics(): Statistics
+intermediateCommit(): void
+isError(): boolean
```

Das Interface DB (in unserem Fall unter org.olat.core.commons.persistence) stellt verschiedene Methoden zur Verfügung, welche die Kommunikation mit der Datenbank und mit den persistenten Objekten erleichtert. Das Interface wird in der Klasse org.olat.core.commons.persistence.DBImpl implementiert und definiert.

Welche Schritte sind erforderlich, um eine Erweiterung an diesem Punkt aufzusetzen?

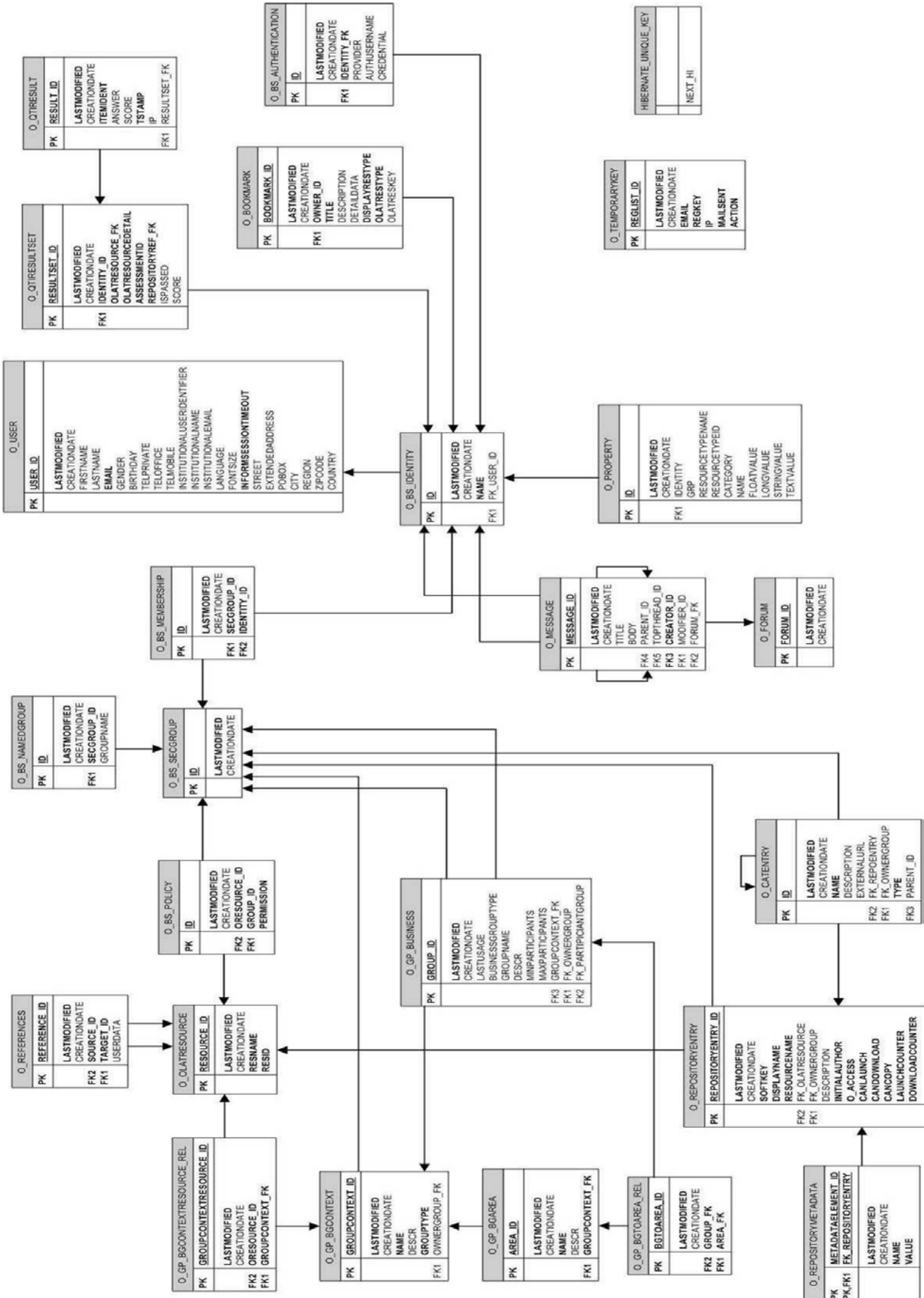
Es muss eine persistente Klasse erstellt werden, die Informationen aus der Applikationsschicht (z.B Benutzerdaten) enthalten soll. Dafür muss diese Klasse von der abstrakten Klasse PersistentObject erben. Anschließend muss eine Hibernate-Mapping-Datei (extension.hbm.xml) erstellt werden, in der die allgemeinen Speicherstrukturen dieses Objekts deklariert sind (also in welche Tabelle und wie die Daten gespeichert werden sollen). Dafür erzeugt man gegebenenfalls neue Tabellen in der Datenbank.
Nun ist es notwendig eine entsprechende Manager-Klasse (Manager) und die Implementierung (ManagerImpl) dieser Klasse zu erzeugen. Mit Hilfe der ManagerImpl ist es dann möglich auf die Datenbankebene zuzugreifen und mit den persistenten Objekten zu arbeiten. Dafür wird innerhalb der ManagerImpl die Klasse DBFactory aufgerufen (DBFactory.getInstance()). Diese gibt eine Instanz der Klasse DBImpl zurück, welche wiederum das Interface DB implementiert.

Ein Beispiel für die Implementierung ist unter anderem in dem Packet org.olat.user realisiert

Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1
Projektleiter: Martin Walther
Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther



Aufgabe 5 16.05.2011

Softwaretechnik-Praktikum SS 2011

swp11-1

Projektleiter: Martin Walther

Bearbeitet von: Matthias Zigan, Johann Berger, Martin Walther