Gruppe: swp10-2 Verantwortliche: Frank Stumpf, Martin Strack, Fabian Externbrink Datum: 16.05.2010

Entwurfsbeschreibung
OLAT

Inhaltsverzeichnis

1. Allgemein

OLAT (Online Learning And Teaching) ist ein webbasiertes Learning Management System mit einer großen Anzahl vielseitiger Funktionen. Das zentrale Thema ist die Kollaboration in Lern- und Arbeitsgruppen, sowie Kursen. Diese können leicht verwaltet und durch verschiedenste Komponenten (Wiki, Forum, Übungen, etc.) erweitert werden.
Darüber hinaus gibt es auch kursunabhängige und kursübergreifende Funktionen. Hierzu zählen eine allgemeine Verwaltung von Lernressourcen sowie die Bereitstellung von Editorenwerkzeuge für Tests, Fragebögen oder Kurse.

Diese Entwurfsbeschreibung befasst sich mit dem Aufbau und der Abwicklung von Operationen auf externe Datenquellen und wie sowohl der Aufruf und die Bearbeitung, als auch der direkte Zugriff auf die Daten vom System getrennt wird.

2. Produktübersicht

Das OLAT besteht aus drei großen Funktionsbereichen.

Der **private Bereich** dient dem Nutzer dazu, sein OLAT nach eigenen Vorstellungen und in eine für ihn produktive Umgebung zu gestalten. Dieser Bereich wird vor allem durch den Tab *Home* repräsentiert.

Der **kollaborative Bereich** dient der Kommunikation zwischen den Benutzern und dem Austausch, sowie der Präsentation von Informationen. Außerdem bietet er Funktionen für eine übersichtliche Organisation von Gruppen:

Der **öffentliche Bereich** dient der Administration zur Verwaltung von Gruppen und einzelnen Nutzern.

Die Funktionalität der einzelnen Bereiche ist abhängig von den Rechten die der Nutzer in einer Gruppe hat. Er kann dabei die System-Rollen Gast, Benutzer, Autor, Gruppenverwalter, Benutzerverwalter und Administrator einnehmen.

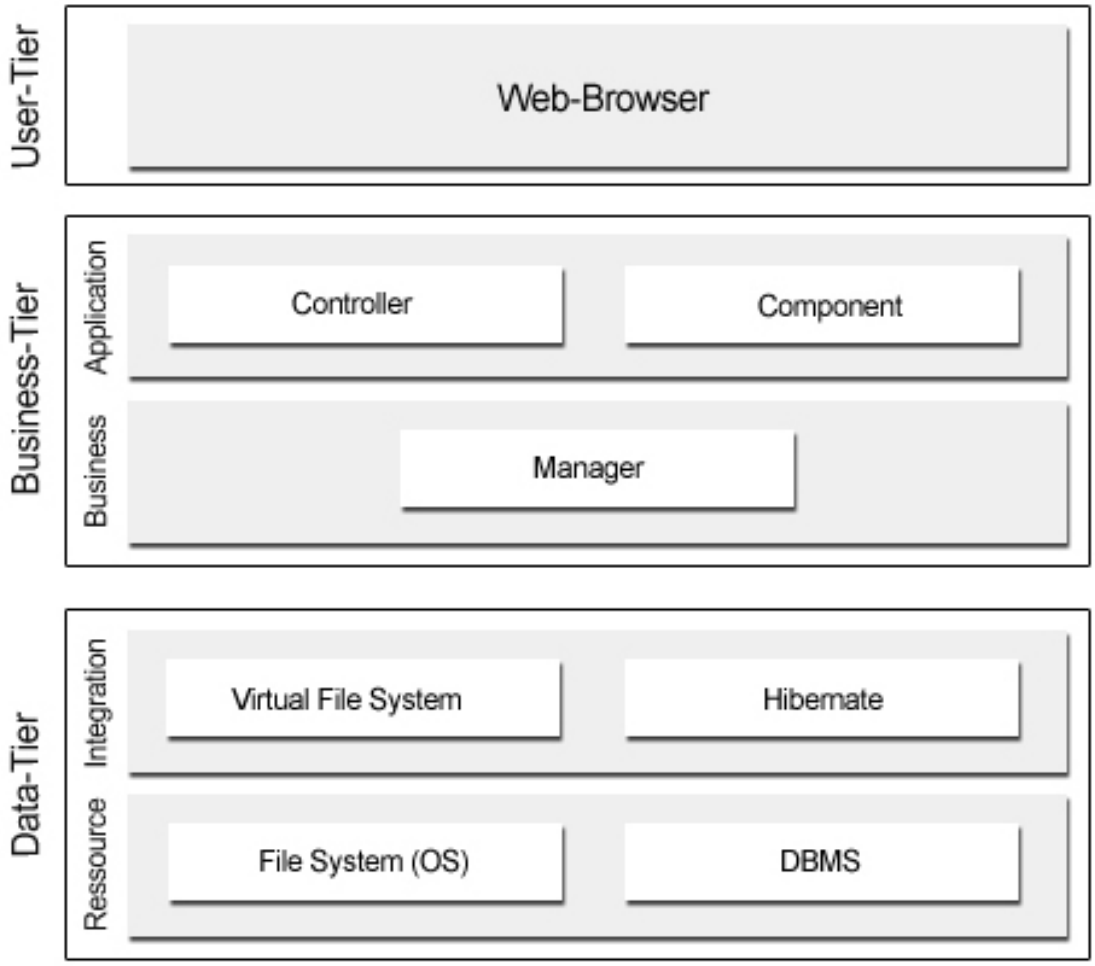
Ein wichtiger Punkt in Bezug auf Datenquellen ist die Speicherung der Datenobjekte.
Alle Inhalte, die OLAT bereitstellt oder durch einen Nutzer hinzugefügt werden, müssen dauerhaft auf dem Server gespeichert und abrufbar sein. Allgemein sind das Objekte wie Nutzer, Kurse, Tests, Foren oder Wikis. Doch auch nutzer- oder gruppenbezogene Dateien (z.B. Dokumente oder Archive) können verwaltet werden.
Zur Verwaltung der Daten hat OLAT die Möglichkeit zur Speicherung im vorliegenden Dateisystem oder die Verwendung einer Datenbank.

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

3.1 Architekturübersicht

Im OLAT werden mehrere Architekturprinzipien angewendet. Das Grundgerüst setzt hierbei auf die Client-Server-Architektur und auf eine 3-Schichten-Architektur, wobei zwischen User-, Business- und Data-Tier unterschieden wird.
Der Aufbau von OLAT ist sehr modular, wodurch eine größtmögliche Trennung der Daten-, Anwendungs- und Darstellungsschicht erreicht wird.
Der Betrieb einer OLAT Instanz wird mit dem Servlet Container Apache Tomcat, sowie einer SQL Datenbank durchgeführt.
Für eine leichte Erweiterung des Systems, sorgen Extension-Points (Erweiterungspunkte). Mit ihnen lassen sich beispielsweise Inhalte in Form von Tabs hinzufügen, ohne den OLAT-Core selbst zu verändern.

OLAT lässt sich nach dem Java-EE Standard in drei grundlegende und fünf detaillierte Schichten einteilen:



3.1.1 Data Tier

Die Datenschicht teilt sich in die beiden Bereiche **Ressource** und **Integration**.

Ressource
Der Grundbaustein des Schichtenmodells ist die **Ressource** Schicht. Dort befinden sich alle zu speichernden Daten, entweder in Form von Dateien im Dateisystem des OS oder in einem DBMS.

Integration
Darauf aufbauend und zur Kapselung der Datenschicht, kommt die **Integration** Schicht. Diese sorgt für einen einheitlichen Zugriff auf die gespeicherten Daten und bietet eine Schnittstelle für das Business-Tier. Den Dateizugriff übernimmt dabei das VFS (Virtual File System) und die Interaktion mit dem DBMS wird über Hibernate realisiert.

3.1.2 Business-Tier

Die Businessschicht teilt sich in die beiden Bereiche **Business** und **Application**.

Business
Die Kommunikation mit der Integrationsschicht übernimmt die **Business** Schicht mittels so genannten *Managern.* Sie verwalten bestimmte Mengen von Datenobjekten und dienen so als indirekte Schnittstelle zu den Daten. Die Objekte können z.B. Nutzer, Kurse, Foren oder Ordner sein. Mit ihnen kann man neue Objekte erstellen, auf Existierende zugreifen oder löschen.

Application
Die nachfolgende **Application** Schicht hat die Funktionalitäten der Applikationslogik und stellt die anzuzeigenden Inhalte zusammen und wandelt diese in ein Format um, welches der Client interpretieren und darstellen kann. Diese Umwandlung erfolgt über Velocity Container und ist wichtig für eine strikte Trennung nach dem MVC Prinzip.

Die hier zum Einsatz kommenden *Controller* bilden zusammen mit den *Components*, basierend auf *Model* und *View*, das MVC-Konzept. Die Controller dienen als Schnittstelle zwischen *Components* und den *Manager* Klassen aus der Business Schicht.

Für Benutzeranfragen aus der darüber liegenden Schicht (User-Tier) ist das *OLATServlet* als Haupteingangspunkt zuständig, welche dann mit Hilfe von *Dispatcher* weiter verarbeitet und weitergeleitet werden.

3.1.3 User-Tier

Abschließend ist die **User-Tier** (Benutzerschicht). Der Webbrowser des Clients sendet Anfragen des Nutzers oder empfängt Daten und stellt diese dar. Er ist also von jeglicher Logik entzogen, so wie es das Java-EE Schichtenmodell vorsieht.

3.2 Hibernate und Velocity

Um die einzelnen Schichten weitestgehend zu entkoppeln, werden in OLAT einige Standards und Frameworks eingesetzt. In der Integrationsschicht wird das **Hibernate Relational Persistence Framework** verwendet. Es dient der Speicherung von Objekten in relationalen Datenbanken und erlaubt so den Austausch des DBMS. Bei der Verwendung muss darauf geachtet werden, dass die Struktur von Hibernate eingehalten wird.
In der Application-Schicht findet **Velocity** Anwendung und setzt dabei das MVC-Prinzip ein. Die Velocity Container ermöglichen das Rendern von HTML Komponenten für die Darstellung des Seiteninhaltes. Die hierfür eingesetzten *Component* Objekte werden durch die Controller gesteuert und über Velocity und den Browser angezeigt.

3.3 Erweiterungspunkt für eine neue Datenquelle

Ein Erweiterungspunkt müsste auf der Datenschicht, genauer der Integrationsschicht, die Daten zuerst von der Quelle laden. Je nach Format der Daten, müssen diese durch einen entsprechenden Parser in ein für die Businessschicht verständliches und brauchbares Format gebracht werden.
Die Datenstruktur kann nun mit Hilfe des Erweiterungspunktes in die Datenbank des OLATs geschrieben werden.
Für die grafische Darstellung des Inhaltes ist dann wieder eine GUI-Erweiterung nötig.

Die Anbindung einer OD Datenquelle an OLAT würde also aus drei Komponenten bestehen:
1. Ein Parser der die Daten aus OD liest und aufbereitet.

2. Ein auf dem Datenbank Erweiterungspunkt aufbauende Anbindung.
3. Eine GUI-Erweiterung, welche die Daten aus der DB darstellt.

3.4 Zugriff auf externe Daten

Die von der Datenschicht bereitgestellten Datenobjekte werden mit dem `OLATRescourceable` Interface eindeutig durch eine ID identifiziert. Somit wird ein leichtes wieder finden über die darüber liegenden Manager ermöglicht.
Objekte welche für die Einlagerung in der Datenbank vorgesehen sind, müssen von der Klasse org.olat.core.commons.persistence.PersistentObject abgeleitet sein.
Um die Daten aus den Quellen zu ermitteln, kann mit `org.olat.core.commons.persistence` über Hibernate auf die Datenbank zugegriffen werden. Dateien innerhalb des Virtual File System werden mit dem `org.olat.core.util.vfs` Interface angesprochen und können wie in Java gewohnt abgerufen werden.

4. Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 org.olat.core.commons.persistence

4.1.1 Interfaces

DB:
Dieses Interface stellt Methoden zur Verfügung, um Hibernate-Sitzungen zu verwalten.

DBJunit:
Dieses Interface stellt eine Methode bereit, die im Testfall alle Daten in der Datenbank löscht.

DBQuery:
Dieses Interface stellt Methoden zur Verfügung, um Datenbank-Anfragen stellen zu können.

ItransactionListener:
Dieser Listener kann zu einer DB-Instanz hinzugefügt werden, um die Befehle „commit“ und „rollback“ zu realisieren.

4.1.2 Klassen

AuditIntercepter:
Diese Klasse dient dem Abfangen von Prüfroutinen.

DatabaseSetup:
Diese Klasse kann dazu genutzt werden, um eine OLAT-Datenbank zu erzeugen.

DBFactory:
Diese Klasse wird genutzt, um eine DB-Instanz zu holen.

DBImpl:
Diese Klasse dient der Verwaltung einer Hibernate-Sitzung.

DBJunitImpl:
Diese Klasse dient dem Testen mit JUnit.

Gruppe: swp10-2 Verantwortliche: Frank Stumpf, Martin Strack, Fabian Externbrink Datum: 16.05.2010

DBModule:
Diese Klasse stellt Methoden zum Initialisieren der Datenbank bereit.

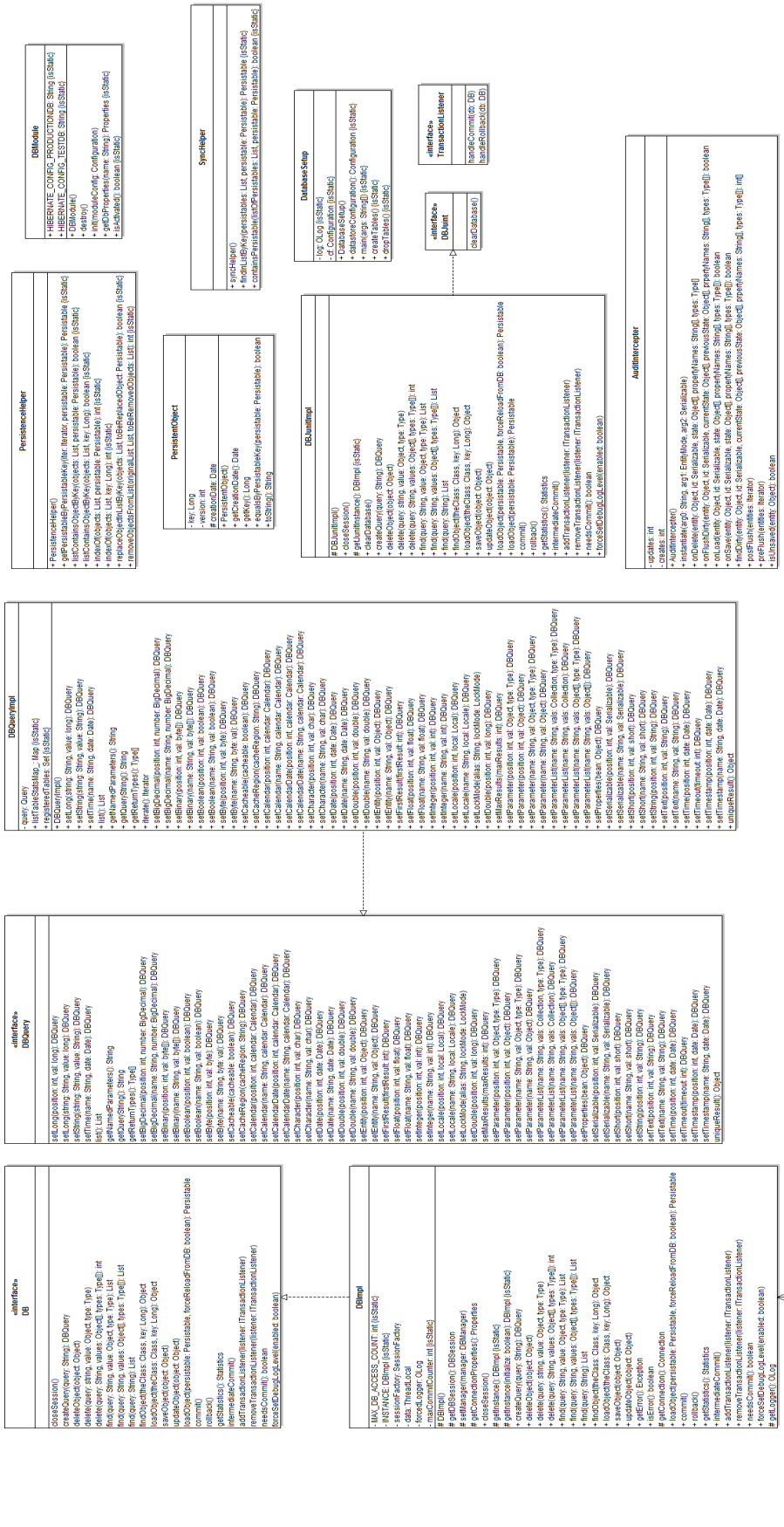
DBQueryImpl:
Diese Klasse ist eine Hülle um eine Hibernate-Anfrage.

PersistenceHelper:
Diese Klasse stellt Hilfsmethoden zur Arbeit mit persistenten Daten bereit.

PersistentObject:
Diese Klasse holt sich den Primärschlüssel und Erstellungsdatum.

SyncHelper:
Diese Klasse ist eine Hilfsklasse zur Synchronisation.

4.1.3 Klassendiagramm



4.2 org.olat.core.util.vfs

4.2.1 Interfaces

OlatRelPathImpl:
Dieses Interface stellt eine Methode zum Holen des relativen Pfades zur Verfügung.

Quota:
Dieses Interface stellt Methoden zur Ermittlung der Quote bereit.

VFSContainer:
Dieses Interface stellt Methoden zur Verwaltung eines Containers mit VFSItems bereit.

VFSItem:
Dieses Interface stellt Methoden zur Verwaltung von VFSItems bereit und liefert Informationen zu diesen.

VFSLeaf:
Dieses Interface stellt eine Datei dar, aus der gelesen werden kann und in die geschrieben werden kann.

4.2.2 Klassen

AbstractVirtualContainer:
Diese Klasse stellt Methoden zur Verwaltung des Containers zur Verfügung.

LocalFileImpl:
Diese Klasse stellt Methoden zur Bearbeitung von Dateien zur Verfügung.

LocalFolderImpl:
Diese Klasse stellt Methoden zur Bearbeitung von Ordnern zur Verfügung.

LocalImpl:
Diese Klasse stellt Methoden zur Verwaltung von VFSItems bereit.

MergeSource:
Diese Klasse ist ein Container, in dem VFSContainer aufgenommen werden können.

NamedContainerImpl:
Diese Klasse dient der Verwaltung von VFSItems im Container.

NamedLeaf:
Diese Klasse umhüllt ein VFSLeaf mit einem anderen Namen.

QuotaManager:
Diese Klasse dient der Verwaltung von Quoten.

StreamedImpl:
Diese Klasse dient der Ein- und Ausgabe.

Gruppe: swp10-2 Verantwortliche: Frank Stumpf, Martin Strack, Fabian Externbrink Datum: 16.05.2010

VFSConstants:
Diese Klasse enthält Attribute, die den VFSStatus angeben.

VFSLeafHandler:
Diese Klasse ist ein Log-Handler, der die Datei für jeden Log-Eintrag öffnet und sofort wieder schließt.

VFSManager:
Diese Klasse stellt essentielle Methoden des VFS bereit.

VFSMediaResource:
Diese Klasse bietet Methoden zur Festlegung von Codierungen an.

VFSStatus:
Diese Klasse bietet einen Konstruktor, um ein VFSStatus anzulegen.

VFSTestMain:
Diese Klasse dient dem Testen.

VirtualContainer:
Diese Klasse dient der Verwaltung von VFSItems im Container.

4.2.3 Klassendiagramm