1. Allgemeines

OLAT (Online Learning and Training) ist ein webbasiertes LMS (Learning Management System) welches zur einfachen Organisation von Lehrveranstaltung genutzt werden kann. Das System bietet die Möglichkeit zur Erstellung und Verwaltung von Kursen in die verschiedene Komponenten eingebunden werden können und einfache Vereinfachung des Lern- und Lehrbetriebs.
OLAT-PA ist eine an der Universität Leipzig entwickelte und eingesetzte Erweiterung für OLAT welche das Erstellen und Verwalten von Prüfungsanmeldungen und ihren Ergebnissen vereinfachen soll.
Das vorliegende Dokument analysiert die grundsätzlichen Entwurfsprinzipien von OLAT und nimmt dabei auch Bezug auf die OLAT-PA Erweiterung

2. Produktübersicht

- OLAT

OLAT bietet ein Kurssystem welches Benutzern das einfache Ein- und Austragen in Kurse sowie deren zugehörige Gruppen ermöglicht. Kursautoren ist es möglich ihre Kurse, Gruppen und Benutzer sowie deren Rechte zu verwalten. Zudem können in jedem Kurs Komponenten wie z.B. Tests, Wikis oder ein Chat eingebunden werden. Ebenso können ganze Kurse importiert werden, was z.B. für Vorlesungsskripte sehr nützlich ist.
OLAT bietet dabei verschiedene Nutzerrollen, den Administrator, verschiedene Autoren und den normalen User. Kursleiter sind hierbei in der Rolle des Autors und können ihre Kurse verwalten und erweitern. Ebenso existiert die Möglichkeit eines Gastzugangs, der jedoch auch in der OLAT-Konfiguration abgestellt werden kann.
Die Verwaltung persönlicher Daten erfolgt für jeden Nutzer in seinem persönlichen Bereich. Dort kann der User auch Kalendereinträge verwalten, Leistungsnachweise einsehen und viele weitere persönliche Einstellungen vornehmen.

- OLAT-PA

Die Erweiterung OLAT-PA gliedert sich grob in zwei Teilbereiche, das Prüfungsverwaltungssystem und die Studentenakte.
Studenten, die die User-Rolle repräsentieren, haben die Möglichkeit eine Studentenakten zu beantragen in der alle wichtige Informationen zu ihren Prüfungsanmeldungen, Ergebnissen und Krankmeldungen verwaltet werden. Des weiteren lassen sich mit dem System alle relevanten Abläufe zu Prüfungsverwaltung, z.B. Anmeldung und Bekanntmachung der Ergebnisse, einfach realisieren.

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Das Business Delegate Prinzip

Business Delegate Prinzip allgemein

Business Delegate ist ein Java-EE-Entwurfsmuster, welches verwendet wird, um die Präsentationsschicht von der Geschäftslogik zu entkoppeln .Details der Implementierung der Geschäftslogik bleiben hierbei für die Präsentationsschicht verborgen.
Der Business Delegate verwendet dann intern die Komponenten der Geschäftslogik und leitet Aufrufe der Präsentationslogik an sie weiter. Bei Änderungen der Präsentationschicht oder Geschäftslogik müssen nun nur noch die Business-Delegate-Funktionen angepasst werden.

zur Umsetzung in OLAT :

Durch die Reaktionen auf Nutzeraktionen wird das Main-Panel mit Inhalten gefüllt bzw. aktualisiert. Allen Komponenten (Darstellungsbausteine, Container) des Panels liegen hierbei Controller-Klassen zugrunde, welche wie Listener agieren. Im Detail bedeutet das, die Identifizierung der entsprechenden Controller erfolgt über die Komponenten, in denen das Event stattgefunden hat.
Je nach Nutzeraktion wird das Panel nun durch die Controller mit neuen Inhalten gefüllt, d.h. Es werden neue Container und Darstellungsbausteine anleget und in das Main-Panel einbaut.
Sollte es hierbei notwendig sein Fragen an das System zu stellen so wird dies vom zentralen Servlet org.olat.servlets.OLATServlet entgegengenommen und weitergeleitet. Welcher Dispatcher sich nun der Anfrage zu widmen hat, wird durch das jeweils ausgelösten Event spezifiziert.

Cross Concerns und Cross Cutting Concerns

Ein weiteres entscheidendes, wenn auch nicht in voller Konsequenz umgesetztes, Designkonzept, ist die Unterteilung der Komponenten in Core Concerns und Cross Cutting Concerns. Core Concerns sind die Kernkomponenten des Systems, welche unabhängig voneinander eine bestimmte Funktion bereitstellen. Cross Cutting Concerns stellen grundlegende Routinen des Systems dar, welche sich aus nichtfunktionalen Anforderungen an das System bilden. In OLAT ist eine genaue Unterteilung schwierig, da wir eher einen symmetrischen Ansatz haben. Trotzdem ist eine Grobe Trennung durchführbar, bei der wir uns vor allem auf die Cross Cuttings konzentrieren wollen.
Zu den Kernkomponenten zählen alle Pakete, deren Funktionen man letztendlich als aktiver Benutzer verwenden bzw. direkt sehen kann . Darunter fällt also beispielsweise das Paket home, welche die Hauptseite darstellt und deren Kontrollfluss verwaltet, genauso die Pakete admin und user und sicherlich auch die Pakete für die Module und Prüfungen. Schwieriger wird es hierbei schon bei den Paketen basesecurity und login, welche den Loginvorgang bereitstellen, hierbei die Rolle des jeweiligen Benutzers überprüfen und ihm eine Sessionrolle zuweisen. Diese Pakete stellen zwar Kernfunktionalität bereit, schneiden sich aber sehr stark selbst und auch andere Komponenten und sind somit schon „crosscutting“.
Ganz klare Cross Cutters sind die Pakete im Ordner core, welche die Grundfunktionen von OLAT bereitstellen. Hier finden wir unter anderem das Paket gui, welche die Basisklassen zur Darstellung aller

grafischen Elemente in OLAT bereitstellt oder das Paket logging, welches von anderen Klassen zur Logfile Erstellung benutzt wird.
Des weiteren findet das Cross Cutting Prinzip Anwendung bei der Klasse Translator. OLAT ist auf Mehrsprachigkeit ausgelegt, weshalb sich in jedem Paket ein Ordner _i18 befindet, welcher die verschiedenen Sprachdateien enthält. Jede Klasse, welche Text ausgibt, benutzt eine Translator Objekt um auf diese Sprachdateien zuzugreifen und die richtige Übersetzung zu auswählen. Durch die dezentrale Lagerung der Sprachdateien für jedes einzelne Paket, können Änderungen schneller durchgeführt und leicht weitere Sprachen für bestimmte Pakete hinzugefügt werden.
Die Dezentralisierung bestimmter Funktionalitäten wurde durch Einzug des Springframeworks weiter fortgeführt. So befindet sich seit OLAT Version 7 in jedem Paket auch ein ordner _spring, welcher eine Datei Context.xml enthält. In dieser können Klassenpfade und weitere Konfigurationen angegeben werden. Früher wurde dies über eine zentrale Stelle geregelt, weshalb Änderungen aufwändiger waren und Extensions nicht so einfach eingefügt werden konnten, wie jetzt.
Durch diese Dezentralisierung entfernt man sich auch immer mehr, wie Eingangs bereits erwähnt, von einer klaren Differenzierung zwischen den verschiedenen Concerns, sondern man hat klare Grundprinzipien, wie das Springkonzept, was aber nicht direkt als crosscutting angesehen werden kann, welche dann von allen Ebenen verwendet werden.

4. Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

Erweiterungspunkte in OLAT

- Liste der Extensionpoints

Erweiterungspunkt: org.olat.gui.control.generic.dtabs.DTabs
Interface: org.olat.extensions.sitescreator.SitesCreator
Beschreibung: Dient zum Anlegen einer neuen Seite als neuen Reiter auf der Hauptseite.
Klasse:

Erweiterungspunkt: org.olat.persistence.DB
Interface: org.olat.extensions.hibernate.HibernateConfigurator
Beschreibung: Erweiterungen für Hibernate Mappings. Die Datenbank kann über diesen Erweiterungspunkt um neue persistente Objekte erweitert werden.
Klasse: org.olat.persistence.DB

Erweiterungspunkt: org.olat.gui.components.Window
Interface: org.olat.extensions.css.CSSIncluder
Beschreibung: Erweiterung um neue, eigene Cascading Style Sheets zur Anpassung der Darstellung.
Klasse: org.olat.gui.css.CSSGenerator

Erweiterungspunkt: org.olat.home.HomeMainController
Interface: org.olat.extensions.action.ActionExtension

Beschreibung: Mit diesem Erweiterungspunkt können weitere Menüpunkte zur Home-Seite hinzugefügt werden. Die Erweiterung enthält Links, Beschreibung und Aktionsbeschreibung.
Klasse:

Erweiterungspunkt: org.olat.dispatcher.DispatcherAction
Interface: org.olat.extensions.globalmapper
Beschreibung: Durch einen neuen Mapper kann ein komplett neuer Arbeitsablauf für einen bestimmten URL-Pfad hinzugefügt werden.
Klasse: org.olat.dispatcher.DispatcherAction

Erweiterungspunkt: org.olat.login.AfterLoginInterceptorController
Interface: org.olat.login.SupportsAfterLoginInterceptor
Beschreibung: Für Module, die direkt nach dem Login arbeiten.
Klasse: org.olat.user.UserModule

- org.olat.home.HomeMainController

Der Erweiterungspunkt *HomeMainController* bindet Action-Extensions ein. Über ihn lässt sich dem Homescreen ein Link mit Beschreibung und ein am Link lauschender Controller hinzufügen - beispielsweise für den ESFLauchController.
Der zu instantiierende Controller sollte eine Subklasse von *BasicController* sein. Um vom *AutoCreator* via Reflection erzeugt werden zu können, muss sein Konstruktor als Argument Referenzen auf *UserRequest* und *WindowControl* übernehmen.

Die Einbindung in OLAT erfolgt über eine xml-Datei mit einem Bezeichner der Form *Context.xml.
Über diese Datei sind dann auch verschiedene Einstellungen möglich.

```
<bean class="org.olat.core.extensions.action.GenericActionExtension" init-method="initExtensionPoints">
   <property name="actionController">
      <bean class="org.olat.core.gui.control.creator.AutoCreator" scope="prototype">
           <property name="className"  value="de.MeinController"/></bean>
        </property>

        [...]

        Scope legt hierbei fest, ob nach dem Singleton-Pattern instantiiert wird oder nicht.

        [...]

        In der Liste wird angegeben, am welchen Erweiterungspunkten unsere ActionExtension
        eingebunden     wird. Eine weitere Möglichkeit wäre SystemAdminMainController.
      <property name="extensionPoints">
              <list>
                      <value>org.olat.home.HomeMainController</value>
              </list>
        </property>
```

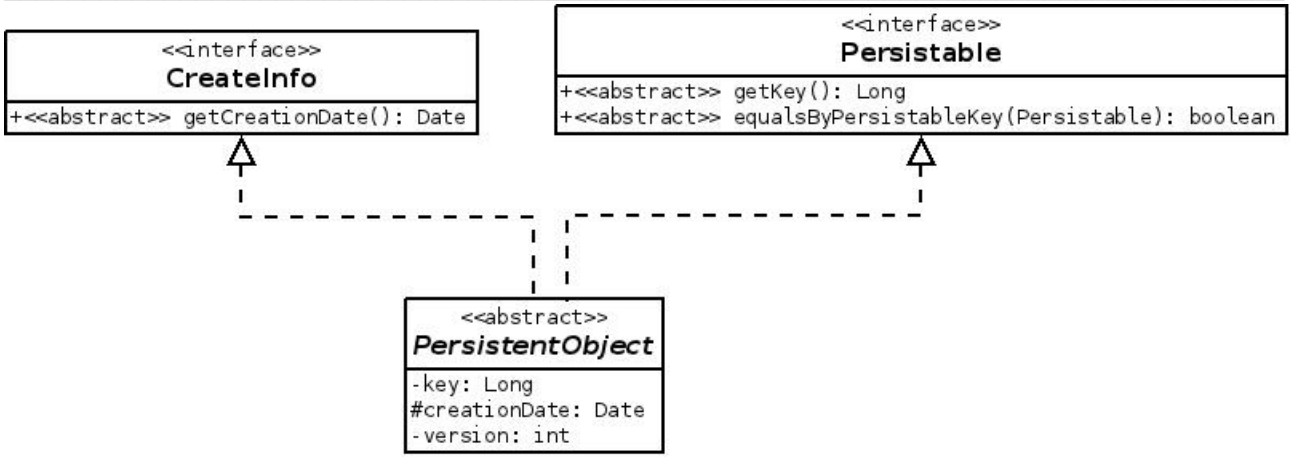
Aktiviert die Erweiterung.
<property name="enabled" value="true" />

- OLAT um Kursbausteine erweitern

Des weiteren besteht die Möglichkeit OLAT um weitere Kursbausteine zu erweitern. Dazu muss eine Klasse XXXCourseNode implementiert werden welches die abstrakte Klasse GenericCourseNode erweitert und die nötigen Controller zum Starten und Editieren bereitstellt. Dabei müssen in jedem Fall die Methoden createEditController, createRunController, isConfigValidund calcAccessAndVisibilty implementiert werden. Im Fall der in OLAT-PA vorgenommen Erweiterung wurde statt GenericCourseNode AbstractAccessableCourseNode(welche selbst wiederum GenericCourseNode erweitert) erweitert.
Zudem ist eine Konfigurationsklasse XXXCouresNodeConfiguration zu realisieren welche das Interface CourseNodeConfiguration implementiert und die Klasse AbstractCourseNodeConfiguration erweitert. Im Fall der OLAT-PA Erweiterung wurde allerdings nicht AbstractCourseNodeConfiguration(gibt es erst seit OLAT Version 7) erweitert sondern das Interface OLATExtension(entfällt in der neueren OLAT Version) implementiert.
Wenn ein Baustein die Möglichkeit bieten soll bewertet werden zu können muss das Interface AssessableCourseNode zur Implementierung verwendet werden. Je nach Umfang und Anforderung an den Kursbaustein muss zudem darauf geachtet ob z.B. eine Datenbankinterkation zu realisieren ist.

Der Extensionpoint persistence.DB

Über den Erweiterungspunkt *org.olat.persistence.DB* können OLAT weitere Hibernate-Mappings hinzugefügt werden.
Damit eine Klasse beziehungsweise ihre Instanzen persistent gespeichert werden kann, muss sie die abstrakte Klasse *PersistentObject* erweitern, welche die Interfaces *CreateInfo* und *Persistable* implementiert. Ein Beispiel hierfür ist die Klasse *org.olat.commons.lifecycle.LifeCycleEntry*.
Diagramm vereinfacht (ohne getter-/setter-Methoden in *PersistentObject*):



Das Attribut *key* steht hierbei für den Primärschlüssel. Das Erstellungsdatum (*creationDate*) und die Version werden von Hibernate benötigt.
Falls die Programmierung hingegen auf ein Interface *IBeispiel* erfolgt, muss *IBeispiel* die Interfaces *CreateInfo* und *Persistable* implementieren. Eine Klasse *IBeispielImpl*, welche *IBeispiel* implementiert, muss dann *PersistentObject* erweitern. Beispielhaft hierfür sind das Interface *org.olat.note.Note* und dessen Implementierung *org.olat.note.NoteImpl*.

Das Mapping selbst wird dann über eine xml-Datei (Bezeichner in der Form *hbm.xml) definiert.

```
<class name="org.olat.Test" table="o_test">
```

Bezeichnet, auf welche Tabelle die Klasse Test gemappt wird. Konvention ist, dass die Tabellen mit „o_“ beginnen.

```
<property  name="creationDate" column="creationdate" type="timestamp" />
```

Mapping eines Attributs. Ist in Java vom Typ Date und wird in der Datenbank auf den Typ timestamp gemappt.

```
<many-to-one name="owner" class="org.olat.basesecurity.IdentityImpl" outer-
join="auto"       cascade="none">
        <column name="owner_id" not-null="false" index="owner_idx"/>
</many-to-one>
```

Mapping einer n:1-Beziehung.

Die zusätzlichen Mappings können der Manager-Instanz durch einen Eintrag in der Datei *databaseCorecontext.xml* bekannt gemacht werden.
Der verwendeten Datenbank müssen dann die zusätzlichen Tabellen für die persistierbaren Klassen hinzugefügt werden.

Zu kritisieren ist an diesem Erweiterungspunkt, dass die Eintragung zusätzlicher xml-Dateien für das Mapping an zentraler Stelle – der Datei *databaseCorecontext.xml* – erfolgen müssen.
Anregung für eine bessere Lösung kann der Erweiterungspunkt *AfterLoginInterceptorController* sein. Hierbei wird dem *AfterLoginInterceptionManager*, welcher als Singleton realisiert ist, für jeden Controller der dem Nutzer nach Login präsentiert werden soll, eine Konfiguration übergeben.
Ein entsprechender *HibernateMappingManager* könnte dann eine Methode besitzen, welche die

zusätzlichen Mappings übernimmt. Durch die Struktur der xml-Dateien bietet sich auch hier eine Wrapper-Klasse für eine List<Map> an.
Ein weiterer Kritikpunkt ist, dass die zu den persistierbaren Objekten gehörigen Tabellen manuell hinzugefügt werden müssen. Arbeitserleichternd wäre, wenn die Managerklasse bei Hinzufügen eines Mappings automatisch ein Skript für den jeweiligen SQL-Dialekt erzeugen würde. Sinnvoll wäre hierbei die Realisierung mittels eines Strategy-Patterns.

OLAT-PA fügt in folgenden Klassen Mappings für Hibernate hinzu :
- de.xman.appointment.AppointmentImpl
- de.xman.comment.CommentImpl
- de.xman.comment.CommentEntry
- de.xman.esf.ElectronicStudentFileImpl
- de.xman.exam.ExamImpl
- de.xman.illness.IllnessReportEntry
- de.xman.illness.IllnessReportImpl
- de.xman.module.ModuleImpl
- de.xman.protocol.ProtocolImpl
- de.xman.studypath.StudyPathImpl