Entwurfsbeschreibung des Prototypen

der Gruppe swp10-1

1. Allgemeines

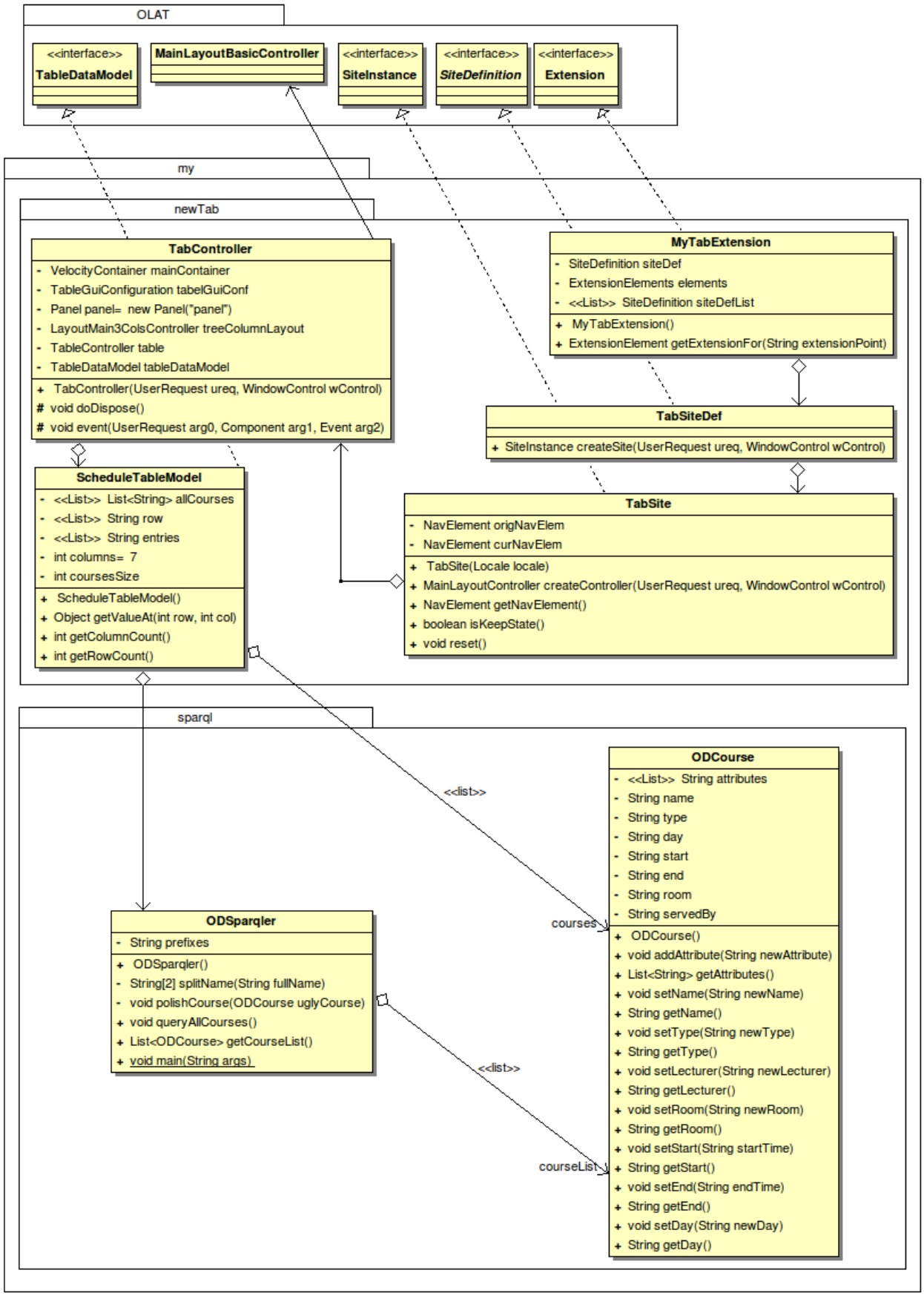
Unser Produkt ist eine in Java geschriebene Erweiterung der OLAT-Lernplattform, Version 6.2.2. Es liest Informationen über Lehrveranstaltungen aus der Datenquelle der Universität Leipzig (im folgenden Datenquelle) und stellt sie dem Nutzer in OLAT zur Verfügung.

2. Produktübersicht

Unser Produkt erfragt alle Lehrveranstaltungen und zugehörige Veranstaltungsdaten des Instituts für Mathematik und Informatik der Universität Leipzig aus der zu diesem Zweck angelegten Datenquelle und stellt diese innerhalb der Lernplattform OLAT als Tabelle dar. Die Abfrage aus der RDF-basierten Datenquelle erfolgt im Prototyp noch direkt bei Aufruf des Tabs „Stundenplan“. Die Tabelle enthält momentan Titel, Typ, vortragenden Dozenten, Wochentag, Start- und Endzeit sowie den Raum der Lehrveranstaltungen. Der Nutzer kann die Spalten der Liste beliebig anordnen, sie nach jedem Merkmal sortieren oder sie auch als Microsoft® Excel-Tabelle herunterladen.

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Um sich einen Überblick über die Struktur unseres Produktes zu verschaffen, sollte man zunächst das Klassendiagramm auf der nächsten Seite betrachten.
Wie man sieht, unterteilt sich unser Produkt, das „my“-Paket, in die Pakete „newTab“ und „sparql“. Ersteres regelt die Darstellung und Funktionalität im OLAT und letzteres realisiert die Abfrage aus der Datenquelle.
Um eine Erweiterung in OLAT einzubinden, orientierten wir uns an der vorgegebenen MVC-Architektur, wobei der View ein velocity-container ist, der die Javaanweisungen in html übersetzt und in den Browser rendert. Für den Controller stellt OLAT diverse abstrakte Klassen mit unterschiedlicher Funktionalität zur Verfügung, welche dann erweitert werden. Hierzu gehören Controller für die 3-Spalten-Darstellung im OLAT, Tabellen, Formulare, etc. Man kann seinen Controller einem bereits bestehenden Navigationselement (Tab) zuordnen oder mittels Implementierung verschiedener Interfaces ein Eigenes erstellen. Hierbei kann man die Sichtbarkeit für die unterschiedlichen Nutzergruppen regeln und die Instanzen dem jeweiligen Profil des aufrufenden Nutzers anpassen.


OLAT
<<interface>>
TableDataModel
MainLayoutBasicController
<<interface>>
SiteInstance
<<interface>>
SiteDefinition
<<interface>>
Extension
my
newTab
TabController
- VelocityContainer mainContainer
- TableGuiConfiguration tabelGuiConf
- Panel panel= new Panel("panel")
- LayoutMain3ColsController treeColumnLayout
- TableController table
- TableDataModel tableDataModel
+ TabController(UserRequest ureq, WindowControl wControl)
# void doDispose()
# void event(UserRequest arg0, Component arg1, Event arg2)
MyTabExtension
- SiteDefinition siteDef
- ExtensionElements elements
- <<List>> SiteDefinition siteDefList
+ MyTabExtension()
+ ExtensionElement getExtensionFor(String extensionPoint)
TabSiteDef
+ SiteInstance createSite(UserRequest ureq, WindowControl wControl)
ScheduleTableModel
- <<List>> List<String> allCourses
- <<List>> String row
- <<List>> String entries
- int columns= 7
- int coursesSize
+ ScheduleTableModel()
+ Object getValueAt(int row, int col)
+ int getColumnCount()
+ int getRowCount()
TabSite
- NavElement origNavElem
- NavElement curNavElem
+ TabSite(Locale locale)
+ MainLayoutController createController(UserRequest ureq, WindowControl wControl)
+ NavElement getNavElement()
+ boolean isKeepState()
+ void reset()
sparql
<<list>>
courses
ODCourse
- <<List>> String attributes
- String name
- String type
- String day
- String start
- String end
- String room
- String servedBy
+ ODCourse()
+ void addAttribute(String newAttribute)
+ List<String> getAttributes()
+ void setName(String newName)
+ String getName()
+ void setType(String newType)
+ String getType()
+ void setLecturer(String newLecturer)
+ String getLecturer()
+ void setRoom(String newRoom)
+ String getRoom()
+ void setStart(String startTime)
+ String getStart()
+ void setEnd(String endTime)
+ String getEnd()
+ void setDay(String newDay)
+ String getDay()
ODSparqler
- String prefixes
+ ODSparqler()
- String[2] splitName(String fullName)
- void polishCourse(ODCourse uglyCourse)
+ void queryAllCourses()
+ List<ODCourse> getCourseList()
+ void main(String args)
<<list>>
courseList

4. Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1. Das newTab-Paket

Dieses Paket realisiert die Darstellung der Liste von Lehrveranstaltungen in einem Navigationselement (Tab) im OLAT. Um zunächst eine Erweiterung in OLAT einzubinden, muss eine Klasse existieren, welche das vom OLAT bereitgestellte Extension-Interface implementiert. Dies ist nur notwendig, wenn man seine Erweiterung als kompakte jar-Datei einbinden möchte. Im Entwicklermodus kann man seine Erweiterung auch in die Ordnerstruktur von OLAT integrieren und an der für sie vorgesehenen Stelle in die olat_extensions.xml einbinden. Die Extension-Klasse bündelt also jene Informationen.
Um nun einen Tab zu erstellen ist die Implementierung der Interfaces SiteDefinition und SiteInstance vonnöten. Erstere regelt dabei für welche Nutzergruppen die Seite sichtbar sein soll und gibt die Anzeigeeinstellungen aus dem Nutzerprofil weiter. Im Prototyp wird davon lediglich die Spracheinstellung genutzt, um die Menüführung und Nutzungshinweise (bei Existenz einer Übersetzungsdatei) für die gewählte Sprache in jener darzustellen. Wir haben für den Prototyp eine deutsche und eine englische Übersetzungsdatei angefertigt.
Die SiteInstance-implementierende Klasse (TabSite.java) hingegen erzeugt den eigentlichen Tab sowie eine Arbeitskopie davon. In dieser Klasse wird der Controller (TabController.java) instanziiert, der für das Layout und die Logik hinter dem neuen Tab verantwortlich ist und auch der View in Form eines velocity-containers bereitstellt. Unser Controller erbt vom MainLayoutBasicController, wodurch die Integration ins das Übersetzungssystem von OLAT gewährleistet, sowie das Event Management ermöglicht wird.
Um das OLAT spezifische Look and Feel zu realisieren, instanziieren wir den LayoutMain3ColsController, der alle Elemente der Seite in einem auf drei Spalten basierenden Layout darstellt. Diese werden im OLAT meist für einen zum Tab gehörigen Menübaum(links), den Inhalt(mitte) und eine Schnellsprungliste(rechts) genutzt. Wir nutzen im Prototyp nur die mittlere Spalte, da keine weiteren Unterseiten vorhanden sind.
Als letztes instanziieren wir noch einen von OLAT bereitgestellten Tabellencontroller und übergeben ihm unser eigenes Tabellendatenmodell (ScheduleTableModel). Dieses erzeugt mit Hilfe des sparql Paketes die jeweiligen Zelleninhalte und baut die komplette Tabelle auf.

4.2. Das sparql-Paket

Dieses Paket realisiert die Abfrage der Lehrveranstaltungen aus der Datenquelle und stellt eine Datenstruktur zur weiteren Verarbeitung zur Verfügung. Die Klasse ODCourse stellt die Datenstruktur dar, welche pro Instanz jeweils eine Lehrveranstaltung speichert und die zum Abruf der Daten erforderlichen getter- und setter-Methoden zur Verfügung stellt.
Die Klasse ODSparqler nutzt die JenaAPI, um die Abfragesprache SPARQL für RDF-Datenquellen in den Javaquelltext einzubetten. Zur Nutzung der JenaAPI werden deren Bibliotheken benötigt. Zum jetzigen Zeitpunkt ist das Präfix und die später abrufbare Liste von Lehrveranstaltungen als Klassenvariable gespeichert. Die Abfrage der Datenquelle geschieht über die Methode queryAllCourses(), die auch das benötigte SPARQL Query als String enthält. Nach der Anfrage werden die Ergebnisse von eventuellen Formatierungsunsauberheiten gereinigt und als Liste vom Typ ODCourse gespeichert. Diese Liste der Anfrageergebnisse kann dann als ArrayList<ODCourse> durch getCourseList() abgerufen werden.