Entwurfsbeschreibung der Softwarestudie

1. Allgemein

In dieser Aufgabe sollte ein Prototyp erstellt werden. Für unsere Projekt-Gruppe bedeutet dies eine Komponente für das OntoWiki zu entwickeln, welche, kurz erklärt, eine neue OntoWiki Service URI zur Verfügung stellt ([OntoWiki]/syncml/] und darüber Informationen über die verfügbaren Model le des angemeldeten Nutzers (HTTP Auth) ausgibt.
Dabei wurden folgende Fragen kurz erklärt, um die Grundlage für den Prototypen zu schaffen:

Was ist der Unterschied zwischen den Erweiterungstypen?

Komponente: die Komponente ist der Teil der Software die zusammen wirken um ganze Systeme zu formen und wird direkt oder indirekt von einer Systemklasse abgeleitet. Alle Komponenten sind verkapselt und verpackt.

Plug-In: das Plug-In ist die kleinste Art der Erweiterung. Stellt eine bestimmte Zusatz- oder Hilfsfunktion bereit.

Modul: Module sind in sich geschlossene PHP-Skripte, die es ermöglichen, an einer bestimmten Position im Template etwas auszugeben.

Wie greife ich auf die aktuelle Wissensbasis zu?

Die vorhandene Datenbasis wird mit der SPARQL-Sprache abgefragt. Die SPARQL-Anfragen werden mit der SPARQL-Query-Methode bearbeitet und es werden Arrays oder boolsche Werte zurückgegeben. Mit Hilfe der addStatement()-Funktion werden neu Triple hinzugefügt.

Wie greife ich auf die aktuelle Ressource zu?

Dafür sind die Controller der MCV Framework zuständig. Denn diese sind z.B. verantwortlich für die Aktionen der verschiedenen View-Tabs auf der OntoWiki-Oberfläche (Ressource-Controller). Aufrufe von Controller-Methoden werden über URI ausgeführt. Z.B.: http://.../ontowiki/resource/view/Max, hier soll auf den Ressource-Controller mit der Aktion „view“ und der Ressource „Max“ zugegriffen werden.

Wie stelle ich eine SPARQL-Anfrage?

Man wählt im Menü „Extras2 und danach „SPARQL Query Editor“, wodurch ein Textfenster geöffnet wird in der wir nun eine SPAQRL-Anfrage stellen können.
Eine einfache Anfrage wäre z.B.:

```
PREFIX ex:<.....>
SELECT ?title
FROM <http://....)>
WHERE {
        {?buch ex:VerlegtBei ?<http://springer.com/Verlag>.}
        UNION(OPTIONAL) {?buch   ex:Titel   ?title.
        FILTER(langMATCHES(Lang(?text),“de“))}
        }ORDER BY ?title
```

Sollte im System schon die Knowledge Bases (Wissenbasis) importiert sein, brauchen wir nun nur

noch auf Submit klicken.

Eine weiter Möglichkeit SPARQL-Anfragen zu stellen besteht darin das man eine Instance Klasse Erfurt_Sparql_Endpoint_Default erzeugt und die entsprechenden Parameter dadurch setzt indem man die Funktionen setQuery(....),addModel(...),setRenderer(...) aufruft. Durch die Ausführung der Funktion query() erhalten wir ein Array mit der Form (head, data, col) zurück, welches uns die benötigten Informationen liefert.

Wie lese ich die private Konfiguration der Erweiterung aus?

Im OntoWiki-Extendsion-Ordner sind unteranderem zwei gewöhnliche Konfigurationsdateien: config.ini und strings.ini. In der config.ini stehen die wichtigsten Werte als Variablen drin, die von der ganzen Software benutzt werden. Die Datei strings.ini stellt die verschiedenen Sprachen zur Verfügung die auf der GUI bzw. auf dem Label, den Buttons usw. angezeigt werden.
Das Einlesen von diesen Variablen ist relativ einfach, in config.ini steht z.B. der Code:
language.gui = english;
Dadurch können wir das folgende Programm ausführen um die Werte einzulesen:

```
$configFile = „path/to/Config.ini“;
$config = new Erfurt_Config($configFile,’private’,true);
Echo $config->language->gui;
```

Dadurch wird dann „english“ angezeigt.
Durch die Technik von Zend Registry können wir im Programm auf die verschiedenen Variablen einfach zugreifen. Als erstes wird dafür der Wert eingesetzt:
Zend_Registry::set(,config’,$cinfig);
und wenn man dann im Quelltext folgende Befehle schreibt:

```
$config=Zend_Registry::get(,config’);
Echo $config->language->gui;
```

wird das gleiche Ergebnis ausgegeben.

Wie erstelle ich ein Template für die Ausgabe und fülle es mit Werten?

Unter dem Ressource Ordner werden drei Templates gestellt: „edit“, „view“ und „list“.
Mit Hilfe von „edit“ können Informationen ausgegeben und geändert werden. Im Gegensatz zu „edit“ werden im „View“ Informationen angezeigt, die aber nicht verändert werden können.
„List“ zeigt alle Informationen mit einer kurzen Beschreibung, die mit entsprechendem Verweis auf den „view“ versehen sind.

Wie gebe ich korrekt Zeichenketten für die GUI aus (I18N)?

I18N steht für Internationalisierung, z.B.: werden hierbei Beschreibungstexte nicht im fest im Quellcode codiert, sondern es müssen dafür Variablen verwendet werden, die von einer Quelle zur Laufzeit eingelesen werden. Aber auch die Datumsformatierungen und die sprachabhängige Oberflächengestaltung gehören hier dazu.

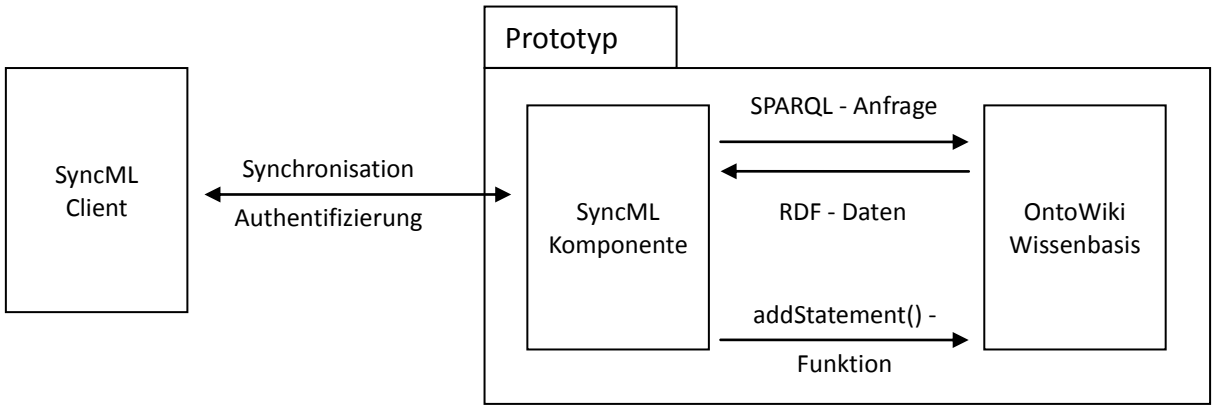
Welche Möglichkeiten gibt es, um Statements zu einer Wissensbasis hinzuzufügen?

Eine Möglichkeit wird durch die API Funktion geboten, die schon unter „Erfurt“ angegeben wird. Zum Beispiel mit Erfurt_Rdfs_Class_Abstract::addInstance(), kann man im API Handbuch vom OntoWiki bzw. Zend_Dom nachschlagen.
Eine weitere Möglichkeit besteht darin einfach selbst eine neue Wissensbasis zu konstruieren. Dies ist jedoch eine sehr aufwendige Methode, da solche Datenbanken sehr groß sind und nicht immer

wieder neu erzeugt werden sollten.

2. Produktübersicht

Das SyncPKM-Projekt verläuft zwischen Client und dem OntoWiki im Hintergrund. Folglich benötigt die reine Synchronisation keine graphische Aufarbeitung im OntoWiki. Aus diesem Grund besitzt auch der Prototyp keine Benutzeroberfläche. Die Funktionsweise ist wie folgt beschrieben:



3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Die Struktur des Prototypen basiert auf der Extensionarchitektur des OntoWiki. Es wurde eine Extension vom Typ Komponente gewählt, da der Prototyp nur einen neuen Dienst bereitstellt und keine GUI Ausgabe verwendet.
OntoWiki wird damit um eine Komponente erweitert, die die Kommunikation mit einem SyncML Server und weiter mit einem SyncML-Client bereitstellt. Dabei wird auf einem vorhanden SyncML Server zurück gegriffen, der ebenfalls eine Extensionarchitektur bereitstellt. Um eine Kommunikation mit der OntoWiki- Komponente zu ermöglichen wird der SyncML-Server um eine Extension erweitert. Diese SyncML Server Extension bzw. die Kommunikation mit dem SyncML-Client ist in dem Prototypen nicht enthalten.

4. Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

Momentan besteht das SyncML-Projekt aus zwei essentiellen Bestandteilen. Aus der OntoWiki- und der SyncML-Server-Extension.
Die *OntoWiki-Extension* basiert auf der Struktur einer Komponente und bietet somit eine ausreichende Schnittstelle zur OntoWiki-Datenbasis. Dabei wird das Zend-Framework benutzt, um die MVC-Architektur des OntoWiki zu verwenden. Des Weiteren werden SPARQL - Anfragen benutzt um Abfragen an die Datenbasis zustellen und die addStatement() - Funktion um neue Tripel hinzuzufügen. Die Authentifizierung am OntoWiki wird über SyncML vorgenommen, wobei die Übertragung mittels HTTP als auch HTTPS möglich sein.
Die *SyncML-Server-Extension* basiert auf der Extensionstruktur des Horde-SyncML-Servers und stellt die Schnittstelle zum SyncML-Server bereit. Da nicht klar ist im welchem Format der Client dem SyncML-Server die Daten sendet, muss sowohl XML als auch WBXML verarbeitet werden können. Weiter Formate werden nicht unterstützt, da nicht bekannt. Für den Fall das eine Synchronisierung fehlerhast abgebrochen wurde, wird eine Funktion bereitgestellt welche alle Datensätze zurücksendet um zu gewährleisten das die Datenbestände konsistent ist. Diese Funktion wird slowSync() genannt.