Entwurfsbeschreibung Vorprojekt

Gliederung:

1. Allgemeines

Unser Projekt ist ein kollaborativer Online-Editor für eine Datenbank zur Erfassung und Pflege von Informationen über deutsch-russische Wissenschaftsbeziehungen im 19. Jahrhundert. Es soll den damit beschäftigten Wissenschaftlern einen einfachen Zugang zu den gespeicherten Daten verschaffen und eine einfache Möglichkeit neue Daten hinzuzufügen bereitstellen. Außerdem sollen Datenbestände einfach verglichen werden können. Schon bestehende Datenbestände sollen dabei integriert werden.

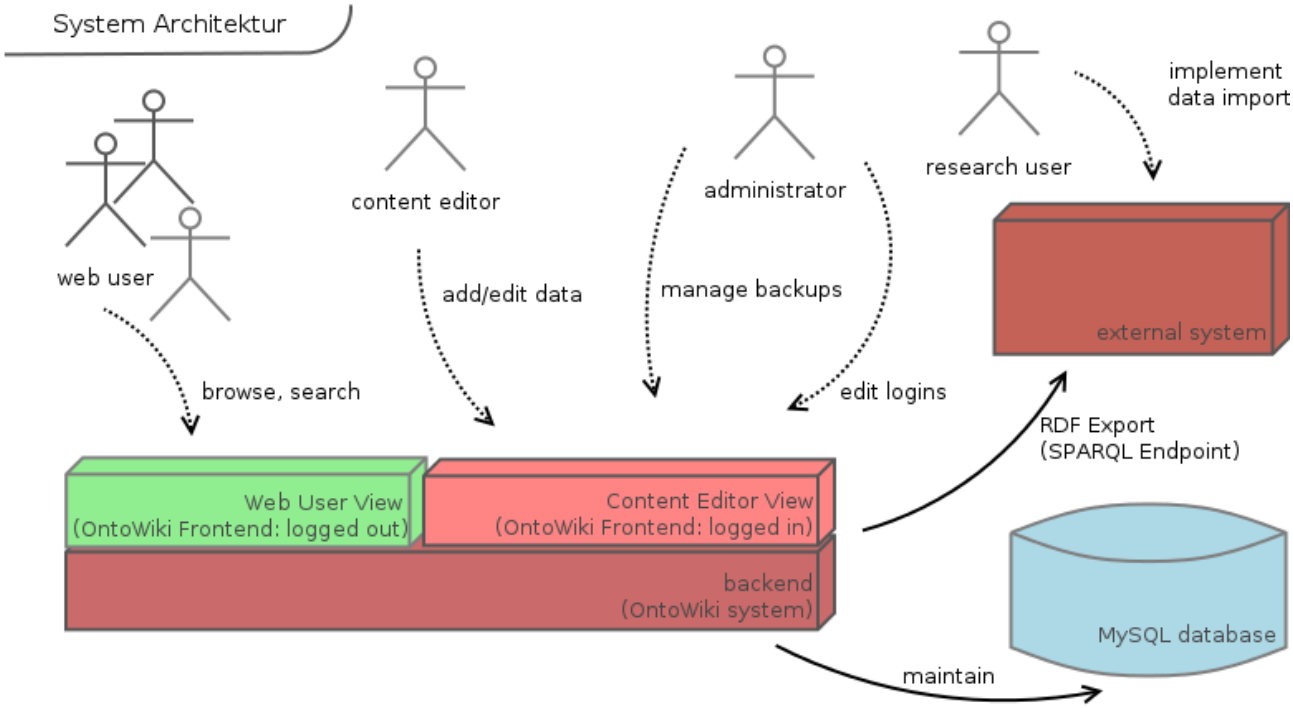
2. Produktübersicht

Nach gründlicher Abwägung der im Recherchebericht besprochenen Vor- und Nachteile der einzelnen Technologien (MySQL, OntoWiki, PHP, SPARQL, ...), haben wir uns serverseitig für das Produkt entschieden, das uns für die Umsetzung am geeignetsten erscheint.

Zusätzlich sind clientseitig Technologien notwendig, die uns bei der Umsetzung gewisser Funktionalitäten unterstützten. Diese werden wir nun im Einzelnen ansprechen. Serverseitig setzen wir OntoWiki ein.

Von der Vielzahl von Komponenten und den durch sie bereitgestellten Funktionen, werden wir uns für unsere Applikation nur einige wenige zu Nutze machen. Durch die Datenbank Komponente binden wir MySQL zwecks Datenhaltung an unsere Anwendung an.

Die GUI (Grafische Benutzeroberfläche), auf die der Benutzer clientseitig zugreifen wird, ist eine auf PHP gestützte HTML Webseite. Der Zugriff auf den SPARQL Endpoint geschieht dabei auch über PHP.



schematische System-Architektur

3. Grundsätzliche Struktur- und Entwurfsprinzipien

3.1 Visualisierung der Inhalte

Die linke Seitenleiste beinhaltet Anmeldeformular und Liste von allen Plug-Ins. Auf dem Hauptfeld daneben erscheinen Ergebnisse der ausgeführten Anfrage. Ganz rechts gibt es die Möglichkeit eine schnelle Suchanfrage zu stellen.

3.2 Erweiterbarkeit

Durch Schnittstellen für Plug-Ins ist das OntoWiki einfach erweiterbar und bietet dem Nutzer eine Möglichkeit, auch auf interne bzw. externe Daten oder Applikationen zuzugreifen.

Unsere Aufgabe beruht auf Erstellung einer Datenbank, in der alle Ressourcen bereits enthalten sind. Auf die Datenbank können jedoch nur die eingeloggten Nutzer zugreifen, die auch über die erforderlichen Rechte verfügen: z.B. Daten hinzufügen, Daten löschen etc.

Das Ziel ist eine Art Filter aufzubauen, der nur registrierten Nutzern alle Informationen anzeigt. Die restlichen User können nur auf einen bestimmten Teil davon zugreifen. Um das zu erreichen, entwickeln wir ein spezielles Plug-In, das eine zweite Datenbank exportiert, indem es die Daten (nach den Zugriffsrechten) filtert.

Die Exportierung der Daten erfolgt dann mithilfe einer eingeloggten Person, die regelmäßig diesen Prozess durchführen muss. Die exportierte Datenbank ist dann für alle Nutzer sichtbar, allerdings ohne Möglichkeit der Veränderung.

4. Struktur der einzelnen Pakete

4.1 Extensions - Struktur

Extension bieten eine flexible Möglichkeit OntoWiki nach belieben zu erweitern. Diese befinden sich im Ordner „extensions“. Darin liegen die verschiedenen Erweiterungen in einzelnen Ordnern. Innerhalb einer Erweiterung befinden sich die Komponentenklassen wie 'Controller.php' und 'Module.php' sowie die 'default.ini'.
Im Ordner 'templates' lassen sich alle Formulare mit der Endung .phtml unterbringen. Sollte der Controllerklasse ein template zugeordnet sein, so muss dieses sich unter 'templates' in einem Ordner mit dem gleichen Namen wie der Controller befinden. Zusätzlich kann ein 'languages'-Ordner erstellt werden, der eine Tabelle mit zu übersetzenden Strings enthält.

4.2 Extensions – Komponenten

4.2.1 default.ini

In der default.ini können grundsätzliche Einstellungen festgelegt werden. So zum Beispiel die Namen der 'templates'- und 'languages' – Ordner. Mit priority und contexts kann die Position bestimmt werden, wobei priority die Reihenfolge festlegt und mit contexts[] = "main.sidewindows" wird bestimmt, dass es sich um eine Erweiterung in der linken Leiste von OntoWiki hält.

4.2.2 Module

Ein Modul ist eine kleine Box in OntoWiki. Erstellt wird es, indem eine neue Klasse angelegt wird, die nach der zu schreibenden Erweiterung benannt ist und als Suffix 'Module' enthält.
Die Klasse muss dann von der von OntoWiki bereitgestellten Modulklasse erben, wodurch sie Methoden zum setzen des Titels und zum Darstellen eines Inhalts erhält.
Dadurch ergibt sich folgendes Grundgerüst:

```
class SearchboxModule extends OntoWiki_Module
{
    public function getContents() {
      // Inhalt des Moduls ist in der searchbox.phtml beschrieben
      $content = $this->render('searchbox');
            return $content;
    }
    public function getTitle(){
      return 'Searchbox';
    }
    public function shouldShow(){
      return true;
    }

}
```

4.2.3 Helper

Mithilfe der Helperklasse kann ein neuer Reiter im Hauptfenster registriert werden.
Das zugehörige Template muss sich innerhalb des 'templates'-Ordner in einem eigenem Ordner befinden, der den gleichen Namen trägt wie die Controllerklasse.

```
class SearchboxHelper extends OntoWiki_Component_Helper
{
    public function init()
    {
      // register new tab 'Searchbox'
      OntoWiki_Navigation::register('searchbox', array(
                'controller' => 'searchbox',     // Controllerclass
                'action'     => 'search',       // action-Method
                'name'       => 'Searchbox ',  // name to display in tab
                'priority'   => 50));         // position
    }
}
```

4.2.4 Controller

In der Controllerklasse befinden sich alle Action-Methoden, die in der Helper- und Modulklasse übergeben wurden.

```
class SearchboxController extends OntoWiki_Controller_Component {
      public function searchAction(){           //    }}
```

Durch $this->getParam('name'); können die Werte, der in die Formulare eingegebenen Boxen, ausgelesen werden. Mit diesen Werten kann dann die SPARQL-Suchanfrage erstellt werden:

```
$query = new Erfurt_Sparql_SimpleQuery();
$query->setProloguePart('SELECT ?name)
      ->setWherePart('WHERE { … }' );
$this->model = $this->_owApp->selectedModel;
$queryResult = $this->model->sparqlQuery($query);
$this->view->queryResult = $queryResult;
```

Das Ergebnis der Anfrage wird wieder um auf den View gesetzt und das entsprechende Template kümmert sich um die Darstellung.

4.2.5 Templates

Mithilfe eines Templates wird das Design der Komponente und deren Inhalt festgelegt.
Das Template für obiges Modul sieht demnach wie folgt aus:

```
<form action="<?php echo $this->actionUrl?>" method="post">
      <p>search string:</p>
      <input name="nameOfPerson" style="min-width:100%;" >
      <p></p>
      <input class="button submit" type="submit" value="start search">
</form>
```

Durch php-Code können vorher festgelegte Werte abgefragt werden.
So muss in diesem Beispiel in der Modulklasse in der getContents() Methode folgendes stehen:

```
$this->view->actionUrl = $url;
```

4.2.6 Language

Damit die angezeigten Strings in der vom Benutzer eingestellten Sprache erscheinen, kann im Ordner 'languages' eine Übersetzungstabelle angelegt werden, die nach der Erweiterung benannt ist und als Suffix die Sprachkennung enthält.
Die Tabelle enthält zwei Spalten, in der linken steht der zu übersetzende String und in der rechten die deutsche Übersetzung.
OntoWiki stellt zum Übersetzen ein translate-Objekt zur Verfügung.

Beispiel: Es existiert die Datei 'searchbox-de.csv', welche folgendes enthält: search; Suche

Ist als Sprache deutsch gewählt, so würde folgender Code dazu führen, dass $string nicht 'search' enthält sondern 'Suche'. $text ist aber weiterhin mit 'result' belegt, da die Sprachtabelle keine Übersetzung dafür anbietet.

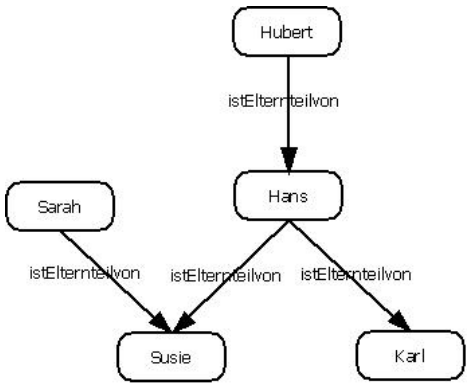
```
$translate        = $this->_owApp->translate;
$string           = $translate->_('search');
$text             = $translate->_('result');
```

5. Datenmodell

Die Datenhaltung von OntoWiki erfolgt vollständig über den sogenannten Triplestore. In dieser Datenbank, die in Form eines MySQL oder eines Virtuoso Servers existieren kann, sind sowohl die von OntoWiki selbst verwendeten Texte und Daten gespeichert, als auch die Datenbestände, die von OntoWiki verwaltet werden. Aufgrund von Performancebedenken entschieden wir uns für die MySQL Variante.

Die Speicherung der Daten in dieser Datenbank wird in der folgenden Grafik schematisch dargestellt:

Der Index ist hierbei eine fortlaufende Nummer, die als Primärschlüssel dient. Die eigentlichen Daten sind als Tripel der Form Subjekt-Prädikat-Objekt gespeichert. Das Subjekt bezeichnet hierbei stets eine Instanz einer Klasse, das Objekt eine andere Klasse oder ein Literal (String, Bool, Int, ...) und das Prädikat die Beziehung zwischen den beiden. Mittels dieser Strukturen können sehr komplexe Beziehungen beschrieben und mithilfe von geeigneten Algorithmen weitere Informationen gewonnen werden. Es folgt der aus den Daten in Grafik 1 konstruierbare Baum.



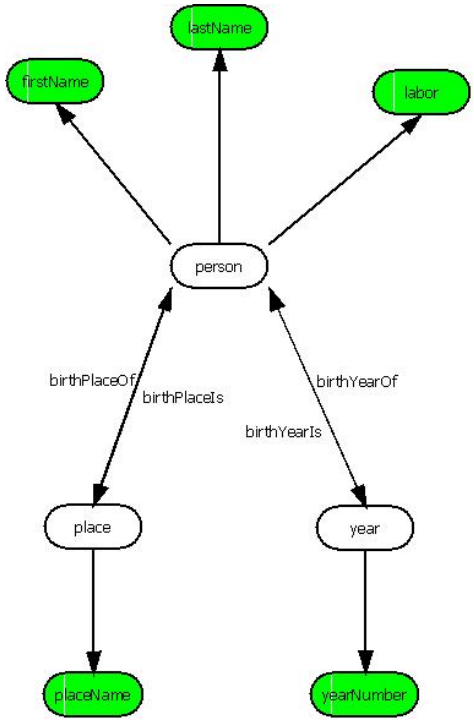
Das Auslesen dieser Beziehungen wird, wie an anderer Stelle erwähnt, vom Erfurt Framework übernommen.

Aus der elementaren Beziehung "ist Elternteil von" kann man nunmehr weitere Informationen entnehmen, z.B. indem man abfragt welche Person Elternteil von einem Elternteil von Susie ist. Diese spezialisierte Form der Abfrage ist mittels der Abfragesprache SPARQL möglich.

Beispiel für SPARQL: (Abfrage aller Personen, die Elternteil von einem Elternteil von Susie sind)

```
SELECT $Großelternteil
WHERE { $Großelternteil abstammung:istElternteilvon Susie}
```

Es ist durch Erweiterungen möglich, Datenbestände sowie Klassenmodelle aus XML Dateien wie RDF, OWL oder Turtle zu importieren oder als solche zu exportieren. Ein Klassenmodell wird dabei als Vokabular bezeichnet, und gibt einen Überblick über alle Objekte sowie über alle Beziehungen zwischen diesen (Vererbung, funktionale Relationen, ...). Für die grafische Darstellung verwenden wir UML-konforme Zeichen. Es folgt eine Grafik unseres für das Vorprojekt entwickelten Vokabulars. Es ist aus Zeit- und Aufwandsgründen natürlich erheblich einfacher als das im Hauptprojekt entwickelte, jedoch eignet es sich bereits gut, um die Funktionalität der Daten zu verdeutlichen.



Die farblosen Flächen sind Klassen von Objekten, denen über die Pfeile, die Object Properties darstellen, andere Objekte und Daten zugeordnet sind. Sie können innerhalb der semantischen Interpretation als Subjekte oder Objekte auftreten.
Die Object Properties treten an die Stelle von Prädikaten, die Instanzen von Klassen miteinander verbinden.
Die grün hinterlegten Flächen stellen Data Properties, also literale Daten, dar. Sie können nur als Objekte im Schema dienen, da sie nicht eindeutig identifizierbar sind. Dennoch sind sie die einzige Möglichkeit wirkliche Daten abzuspeichern.

Durch die inversen Object Properties birthPlaceOf und birthPlaceIs sowie birthYearOf und birthYearIs ist es möglich, ohne erheblichen Mehraufwand Orten die Personen zuzuordnen, die dort geboren wurden oder starben. Hierzu ist es nötig für Orte und Jahre eigene Klassen zu erstellen, wobei die spezifischen Informationen in placeName und yearNumber gespeichert werden.

6. Testkonzept

6.1 Einführung

6.1.1 Wozu Softwaretests?

Um gute und funktionstüchtige Software in einem Projekt zu realisieren ist es ab einem bestimmten maß an Komplexität unausweichlich die einzelnen Komponenten und das Softwaresystem an sich systematisch zu testen. Dabei ist es wichtig auf einen Automatisierten Testablauf zu achten um zum einem die Tests zu optimieren und Fehler in frühen Stadien zu erkennen und zu beheben.
Zur Verwaltung der Tests empfiehlt es sich auf etablierte Test Frameworks wie etwa PHPUnit oder JS-Test zurückzugreifen da diese einen automatisierten Testablauf gewährleisten.

6.1.2 Automatisierung

Um die Tests zu optimieren, bietet es sich an, sie zu automatisieren d.h. das einmal definierte Tests beliebig oft, im Idealfall auch nach Änderungen im Programm, wiederholt werden können. Damit können große Bereiche leichter getestet werden. Etablierte Frameworks helfen dabei automatisierte Testroutinen zu erstellen und zu überblicken.

6.2 Der Testprozess

6.2.1 Komponententests

Komponenten wie etwa Klassen und Methoden werden bei diesen Tests auf ihre Korrektheit und Funktionalität getestet. Dabei werden Testreihen und Beispiele für die einzelnen Klassen und Methoden entwickelt. Dies sollte in der Regel vom Programmierer selbst aus geschehen. Bei jedem Durchlauf eines Tests wird das Ergebnis für evtl. Fehler und Funktionsdefizite sorgfältig dokumentiert. Die Fehlerquelle sollte dann gefunden und wenn möglich behoben werden. Nach erfolgreichen Komponententests ist der Testprozess weiterzuführen.

6.2.2 Integrationstests

In diesem Testabschnitt werden die einzelnen Komponenten auf Zusammenarbeit getestet. Getestet wird nach jeder Woche welche fertiggestellten Module bereits mit anderen zusammenarbeiten können und ob dabei Fehler auftreten bzw. diese kompatibel sind. Werden bei Integrationstests Fehler lokalisiert werden die einzelnen Module angepasst.

6.2.3 Systemtests

Sind Komponenten- und Integrationstests erfolgreich abgeschlossen, so wird eine Vorabversion aus den bestehenden Modulen zusammengestellt. Mit dieser Vorabversion wird aus Nutzersicht geprüft ob die Anforderungen aus dem Lastenheft erfüllt werden. Werden dabei fehlende Funktionalitäten festgestellt sind diese hinzuzufügen. Werden Module dabei verändert, müssen Komponenten- und Integrationstests erneut durchgeführt werden.

6.3 Frameworks

6.3.1 phpUnit

phpUnit ist ein Testframework für php-Skripte und besonders geeignet für einzelne Units und Methoden. Damit also explizit geeignet für Komponententests.

Funktionsweise:

1. Die Tests für eine Klasse ”Class” befinden sich in einer Klasse ”ClassTest”
2. Die Tests sind ”public” Methoden, mit dem Namen test*
3. ClassTest erbt von PHPUnit_Framework_TestCase
4. In den Testmethoden werden assertion-Methoden wie ”assertEquals()” verwendet, um zu überprüfen, dass ein tatsächlicher Wert dem erwarteten Wert entspricht.

Beispiel zum Testen von Array-Operationen:

```
<?php
     class StackTest extends PHPUnit_Framework_TestCase
     {
          public function testPushAndPop()
          {
               $stack = array();
               $this->assertEquals(0, count($stack));

               array_push($stack, 'foo');
               $this->assertEquals('foo', $stack[count($stack)-1]);
               $this->assertEquals(1, count($stack));

               $this->assertEquals('foo', array_pop($stack));
               $this->assertEquals(0, count($stack));
          }
     }
?>
```

Da wir in unserem Projekt das Zend-Framework verwenden welches phpUnit beinhaltet werden wir für automatisierte Tests zum Großteil phpUnit verwenden.

6.3.2 JSUnit

JSUnit ist ein freies Testframework für java-Skript welches nach dem bekannten Testframework Junit konstruiert wurde. Es stellt vor allem Tests auf verschiedenen Umgebungen zurverfügung so z.b. können wir java-skript in verschiedenen Browsern testen oder gar auf verschiedenen Betriebssystemen.

https://github.com/pivotal/jsunit

6.4. Was ist zu Testen?

6.4.1 Plugins

Unter Plugins versteht man, im falle von OntoWiki, kleine aus PHP Dateien aufgebaute Programmteile welche auf „Eventhandling“ ausgelegt sind. Plugins werden wir gebrauchen um nötige Schnittstellen zu erzeugen.

Plugins bestehen wie schon erwähnt aus PHP Dateien die an entsprechender stelle in der OntoWiki MVC Dateistruktur eingefügt werden müssen. Plugins werden wir damit Großteils mit PHPUnit testen. Plugin Tests im eigentlichen Sinn Komponententests werden vom Programmierer selbst erstellt und Dokumentiert. Erst nach erfolgreichem Komponententests werden sie unserem OntoWiki-Repository hinzugefügt und auf Integrationsfähigkeit getestet.

6.4.2 Extensions

Unter Extensions versteht man eine übergeordnete Klasse von Plugins welche auf Funktionsaufrufe und Steuerbarkeit ausgelegt sind. Extensions werden wir benötigen um OntoWiki nach den Kunden wünschen zu Konfigurieren (Einfache GUI, Abfragen und Eintragemöglichkeit in die Datenbank erleichtern).

Ähnlich der Plugins sind Extensions zum Großteil aus PHP Dateien aufgebaut und werden damit auch mit PHPUnit getestet. Extensions werden ebenfalls erst nach erfolgreichem Komponententest in das OntoWiki-Repository eingefügt und auf Integrationsfähigkeit getestet.

6.4.3 XML Validator

Da wir in unserem Projekt mit extrem vielen Daten innerhalb einer .html Datei konfrontiert sind werden wir die Daten dieser .html Datei automatisch generieren lassen. D.h. Wir werden in einer bekannten Programmiersprache einen Skript erstellen um die Daten der .html Datei in XML Form zu bringen. Da die .html Datei teils per Hand geschrieben wurde erwarten wir keine konsistenten Daten. Um diese Konsistenz Fehler schneller zu bearbeiten werden wir XML Validator verwenden um evtl. Syntaxfehler in der XML zu beheben.

7. Glossar

Administrator
Der Administrator verwaltet die Zugänge der Content User und ist für die Datensicherung verantwortlich.

Altdaten
Das sind alle Daten, die bisher erfasst wurden und die in Form der bestehenden HTML-Daten zur Verfügung gestellt werden.

Backend
Das Backend ist der Kern des Systems, dass die Datenverwaltung und Datenhaltung realisiert. Dem Administrator und den Content Editoren werden hier alle sie betreffenden Funktionalitäten zur Verfügung gestellt.

Content Editor
Ein Content Editor hat die Möglichkeit den Datenbestand einer Applikation zu ändern, verfügt also über den Zugang zu dem Backend .

Frontend
Das Frontend ist die Schnittstelle des Systems zu dem Web User. Hier werden die Daten für die Öffentlichkeit visuell dargestellt.

OntoWiki
Das OntoWiki ist ein Wikisystem, in dem alle Daten als Tripel gespeichert werden, was ihre Verknüpfungen ermöglicht. Es ist auch ein Werkzeug, das der kollaborativen Arbeit mehrerer Nutzer dient. Web User können verschiedene Inhalte anlegen und editieren und auch dazu Annotationen bzw. Kommentare schreiben.

SPARQL Endpoint
Durch diese Schnittstelle wird es externen Projekten ermöglicht direkt auf die Daten zuzugreifen.

Vokabular
Ein Vokabular bildet eine Menge von gewählten Bezeichnungen, welche die Datenstruktur in einer Ontologie beschreiben.

Web User
Ein Web User ist ein Nutzer einer Webapplikation, er kann den Datenbestand nicht ändern, sondern die Funktionen einer Applikation nuten z.B.: recherchieren und Suchanfragen stellen.

Extension
Eine Extension dient zum Aufruf von Funktionen in einer Softwareanwendung.

Plugin
Ein Plugin ist ein Softwaremodul, das von einer Softwareanwendung während seiner Laufzeit entdeckt und eingebunden werden kann, um dessen Funktionalität zu erweitern.