Entwurfsbeschreibung

1 Allgemeines

Die entwickelte Software ist eine Weboberfläche mit zugehörigem Backend, welche Suchanfragen gegen eine Ontologie für Morpheme visuell darstellt. Sie dient dazu, für Personen, die nicht mit dem Aufbau von RDF-Datenbanken vertraut sind, die Verwendung der gespeicherten Daten zu vereinfachen bzw. überhaupt zu ermöglichen, ohne sich mit der Thematik von Ontologien zu beschäftigen.

2 Produktübersicht

Die Weboberfläche enthält ein Suchfeld, eine zweispaltige Tabelle zur Darstellung der Suchergebnisse und eine weitere Tabelle für Detailinformationen zu einem ausgewählten Beitrag. Das Suchfeld dient zur Freitextsuche in der Ontologie. Als Ergebnis werden in einer Tabelle alle Morpheme angezeigt, die genau diese orthographische Repräsentation besitzen. Durch Anklicken eines Eintrags in der Ergebnistabelle kann man sich alle weiteren Informationen anzeigen lassen, die zu diesem Morphem in der Datenbank gespeichert sind.
Desweiteren können registrierte Benutzer neue Einträge zur Datenbank hinzuzufügen. Dabei können entweder neue Morpheme erstellt oder noch nicht gesetzte Werte für Eigenschaften eines Morphems ergänzt werden. Außerdem besteht die Möglichkeit Fehler zu melden und Korrekturvorschläge einzureichen.
Der Administrator kann zudem bereits gesetzte Werte für Eigenschaften eines bestimmten Morphems zu ändern. Drüber hinaus existiert eine Administrator-Oberfläche, in der gemeldete Fehler eingesehen und Korrekturvorschläge angenommen oder verworfen werden können.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

Die Entwicklung der Software erfolgte vollständig in Java und als Webapplikation mit einer 3-Schichten Architektur (vgl. Abb. 1). Bewusst haben wir uns auf nur drei grobe Schichten beschränkt, um den Effizienzverlust durch den Transport der Daten über die einzelnen Schichten zu minimieren.

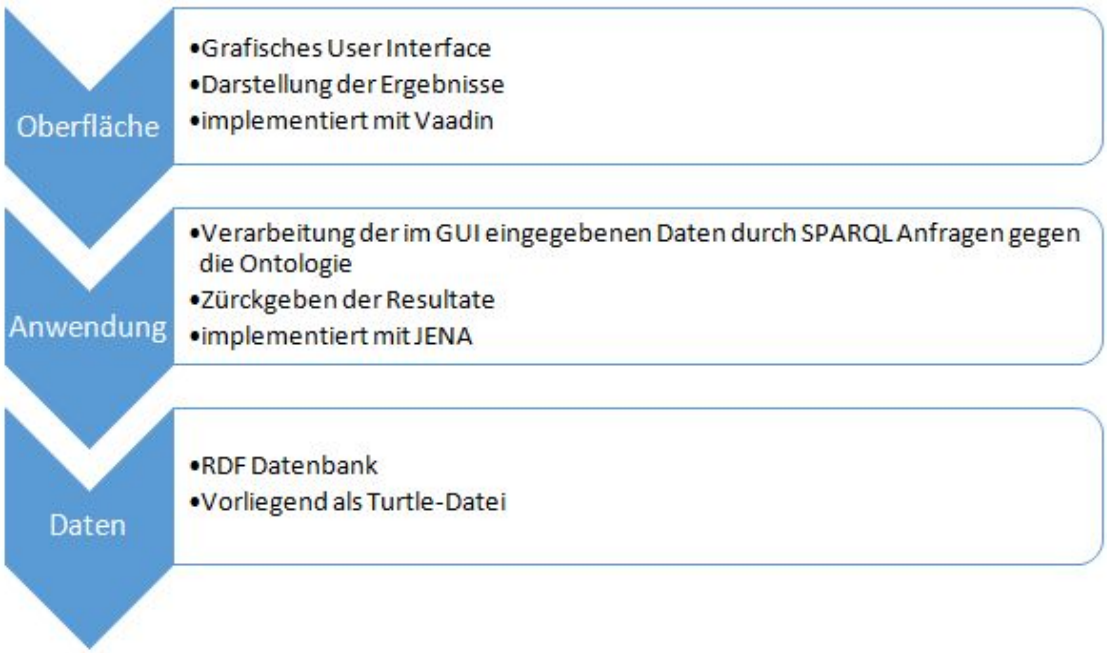

Abb. 1: Grundstruktur der Anwendung.

Als Laufzeitumgebung wurde der Application-Server *Tomcat* verwendet. Die Oberfläche wurde mit *Vaadin* umgesetzt. Sie besteht aus unterschiedlichen Komponenten, die dem Benutzer die verschiedenen Funktionalitäten zugänglich machen. Die Software folgt dem *Model-View-Presenter* Modell. Die Komponenten (Views) implementieren daher keine Funktionalität, sondern dienen lediglich der Eingabe und Darstellung von Daten. Der Presenter vermittelt zwischen den Views und der Morphem-Datenbank (Model). Die Anwendungsschicht wurde mit *Jena* realisiert. Sie liest die Daten ein und erlaubt die Abfrage der Datenbank mittels *SPARQL*. Eine Manipulation der Daten ist ebenfalls möglich. Durch die Persistierung der Datenbank bleiben Änderungen auch nach einem eventuellen Serverabsturz erhalten. Die *ACID*-Eigenschaften werden durch den Aufbau von *Jena* gewährleistet. Die Daten stammen aus dem Projekt *MMoOn*. Sie umfassen eine Ontologie-, eine Schema- und eine Inventory-Datei. Die Daten repräsentieren einen im Turtleformat serialisierten RDF-Graphen. Für die Erstellung wurde ein kleiner Testdatensatz mit deutschen Morphemen zur verfügung gestellt. Die Benutzerverwaltung, Authentifizierung und Autorisierung wurde mit Hilfe des *Shiro* Frameworks umgesetzt.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

4.1 Die Laufzeitumgebung

Die Anwendung wurde mit Java (Version 1.8.0_40) unter Verwendung der Java SE Runtime Environment (Version 1.8.0_40-b27) entwickelt. Als Application-Server wurde die Open Source Implementierung der *Java Servlet, JavaServer Page, Java Expression Language* und *Java WebSocket* Technologien Apache *Tomcat*[1] (Version 8.0.32) verwendet.

[1] http://tomcat.apache.org/

Die Konfiguration der Anwendung erfolgt textdateibasiert. Beim Start des Servers wird die Klasse **`Configuration`** instantiiert, diese liest aus verschiedenen Datein die Pfade zur Datenbank, deren Rohdaten, den Administratornachrichten und die Konfiguration der Benutzer und Zugriffsrechte ein. Dabei wird die Datenbank automatisch initialisiert, falls dies nicht bereits geschehen ist.

4.2 Die Weboberfläche

Die Weboberfläche wurde mit dem *Vaadin*[2] Framework (Version 7.6.5) umgesetzt. Das User Interface besteht aus 4 Views: **`SearchUI`** (Hauptansicht), **`SearchUIAdminPageNew`** (Administrationsoberfläche), **`LogInUI`**, **`LogOut`**.
Den Einstiegspunkt in die Applikation bildet die Klasse **`SearchUI`**. Diese instantiiert das Model, den Presenter und die Komponenten der Benutzeroberfläche. Des Weiteren werden hier die Komponenten in einem gemeinsamen Layout arrangiert.
Jede Komponente wird mittels einer eigenen Klasse und einem Interface realisiert. In der Klasse werden die sichtbaren GUI-Komponenten instantiiert und gelayoutet sowie die benötigten Datenstrukturen realisiert. Die Klasse implementiert das Interface, das die Funktionalitäten der Komponente definiert. Darüber hinaus wird ein Listener-Interface definiert wird, das in dem Interface der Komponente geschachtelt ist. Im Interface der Klasse ist die Methode zum Hinzufügen des Listeners definiert, die durch den Presenter aufgerufen wird um sich selbst als Listener der Komponente zu registrieren.
Der Presenter ist in der Klasse **`SearchPresenter`** realisiert. Die Methoden des Listeners werden durch die Komponentenklassen aufgerufen und rufen selbst wiederum Funktionen der Model-Klasse auf um Anfragen an die Datenbank zu stellen. Desweiteren übergeben sie die Ergebnisse als Parameter von Methodenaufrufen der Komponentenklassen, die dann den View aktualisieren.
In den folgenden Pseudocode-Fragmenten sollen die wesentlichen Strukturmerkmale der verschiedenen Klassen illustriert werden:

```
public interface ComponentInterface {
	public void componentFunction ();

	interface ComponentListener {
		void componentPresenterFunction ();
	}

	public void addComponentListener (ComponentListener listener);
}
```

[2] https://vaadin.com/framework

```
public class Component implements ComponentInterface {
	public void componentFunction () {}

	List<ComponentListener> listeners = new ArrayList<ComponentListener>
	public void addComponentListener (ComponentListener listener) {
		listeners.add(listener);
	}

	public void someFunction () {
		for (ComponentListener listener : listeners) {
			listener.componentPresenterFunction ();
		}
	}
}

public class Model {
	public void modelFunction () {}
}

public class Presenter implements ComponentListener {
	public void componentPresenterFunction () {
		model.modelFunction ();
		component.componentFunction();
	}
}
```

Die Administrator-Oberfläche verfügt über einen eigenen Presenter (**`SearchUIAdminPageNew`**), die Funktionsweise ist analog.

4.3 Die Anwendungsschicht

Die logische Anwendung durchsucht die Datenbank mithilfe der Anfragesprache *SPARQL*. Das Model ist in der Klasse **`QueryDBSPARQL`** implementiert. Die Resultate einer Anfrage werden in einer Liste mit Strings innerhalb der Klasse gespeichert. Die Werte können über Getter-Methoden weiterverwendet werden. Jede neue Suchanfrage leert alle alten Suchergebnisse, damit nicht versehentlich mit veralteten Werten gearbeitet wird. Die Rückgabewerte (Strings) sollten unbedingt vor Weiterverwendung lokal gespeichert werden. Denn für eine entsprechende weitergehende Suche (bspw. alle Informationen zu diesem Morphem) muss genau dieser String an die Funktion übergeben werden. Das liegt in dem Aufbau von RDF-Datenbanken begründet.
Ein String wird durch die Klasse **`SearchField`** übergeben und durch die Methode `searchForSubject(String subject)` der Klasse **`QueryDBSPARQL`** in eine entsprechende *SPARQL*-Anfrage umgewandelt. Diese Anfrage wird dann durch *Jena* auf der Datenbank ausgeführt. Dabei werden die Subjekte von all jenen Tripeln ausgewählt, die der Struktur Subjekt-moon:orthogaphicRepresentation-Objekt entsprechen und deren Objekt gleich dem eingegebenem Text-String ist. Die Liste der Suchergebnisse wird dann durch die Klasse **`Overview`** in einer Tabelle dargestellt.
Die Auswahl einer Zeile in der Tabelle der Suchergebnisse triggert die Methoden `searchForObject(String subject)` und `searchForEmptyProperty(String subject)` der Klasse **`QueryDBSPARQL`**. Dabei werden all jene Prädikate und Objekte von Tripeln ausgewählt, deren Subjekt dem ausgewählten Suchergebniss entspricht bzw. Prädikate die dem Subjekt zuweisen kann, aber noch mit keinem Objekt verknüpft wurden.

Das Ergebnis dieser Anfrage wird durch die Klasse **`DetailView`** in einer Tabelle von Prädikat-Objekt-Paaren dargestellt.
Eine Manipulation der Daten ist möglich durch das Klicken der geeigneten Buttons in der Oberfläche. Diese rufen dann entweder die Methode `insertValue(String subjectString, String propertyString, String objectString)` oder die `deleteValue(String subjectString, String propertyString, String objectString)` auf. Beide Methoden nehmen die Werte eines Tripels entgegen und fügen dieses nach den in der Datenbank definierten Eigenschaften ein oder löschen es. Ob diese Operation erfolgreich war, kann durch die `verifyUpdate(String subjectString, String propertyString, String objectString)` Methode überprüft werden. Diese gibt einen Boolean-Wert zurück.
Methoden, um die Datenbank in eine *Turtle*-Datei zu schreiben bzw. von einer *Turtle*-Datei einzulesen, sind bereits implementiert und können zum Export bzw. zum automatischen erstellen von Backups verwendet werden.

4.4 Die Datenschicht

Zur Speicherung, Verwaltung und Abfrage der Daten wurde das *Jena*[3] Framework (Version 3.0.1) verwendet. Die Rohdaten liegen als serialisierte RDF-Daten im Turtleformat vor (vgl. 5. Datenmodell). Beim ersten Start des Servers wird ein Triplestore aus den serialisierten Daten erzeugt. Auf diesem werden alle Anfragen durchgeführt und Einfügungen und Änderungen persistiert. Die Erstellung von Backups der Datenbank erfolgt ebenfalls in Form von Turtledateien in konfigurierbaren Intervallen. Die komplette Datenschicht ist in der Klasse **`QueryDBSPARQL`** implementiert.

4.5 User Management, Authentifizierung und Autorisierung

Die Benutzerverwaltung und -anmeldung sowie die Rechteverwaltung wurde mittels des *Shiro*[4] Frameworks (Version 1.2.4) implementiert. Beim Start des Servers wird WebEnvironment instantiiert, dass eine Instanz des SecurityManagers umfasst. Das WebEnvironment und der SercurityManager werden mittels einer Textdatei konfiguriert. Auf diese Weise werden Benutzernamen und Passwörter bereitgestellt, den Nutzern werden Rollen zugewiesen, denen wiederum Rechte zugewiesen sind. Die programmatische Abfrage von Benutzerrolle, Rechten und Authentifizierungsstatus wird zur Anpassung des Layouts genutzt, um so Funktionen für entsprechend berechtigte Benutzer zugänglich zu machen. Die textbasierte Konfiguration erlaubt keine dynamische Benutzerverwaltung, was allerdings aufgrund der stark begrenzten Zahl zu erwartender Benutzer zunächst vertretbar ist.
Das WebEnvironment filtert alle Anfragen an den Server und erlaubt auf URLs bezogene User-abhängige Aktionen. Zugriffe auf die Administratorseite durch unauthentifizierte Benutzer werden auf eine Login-Seite umgeleitet, die durch die Klasse **`LogIn`** implementiert ist. Das Logout wird mittels der Klasse **`LogOut`** implementiert. Die Klasse ist auf die URL /logout gemappt, auf die gleichzeitig ein Filter angewendet wird, der nachdem Logout eine Weiterleitung zur Hauptseite erlaubt.

[3] https://jena.apache.org/
[4] http://shiro.apache.org/

5 Datenmodell

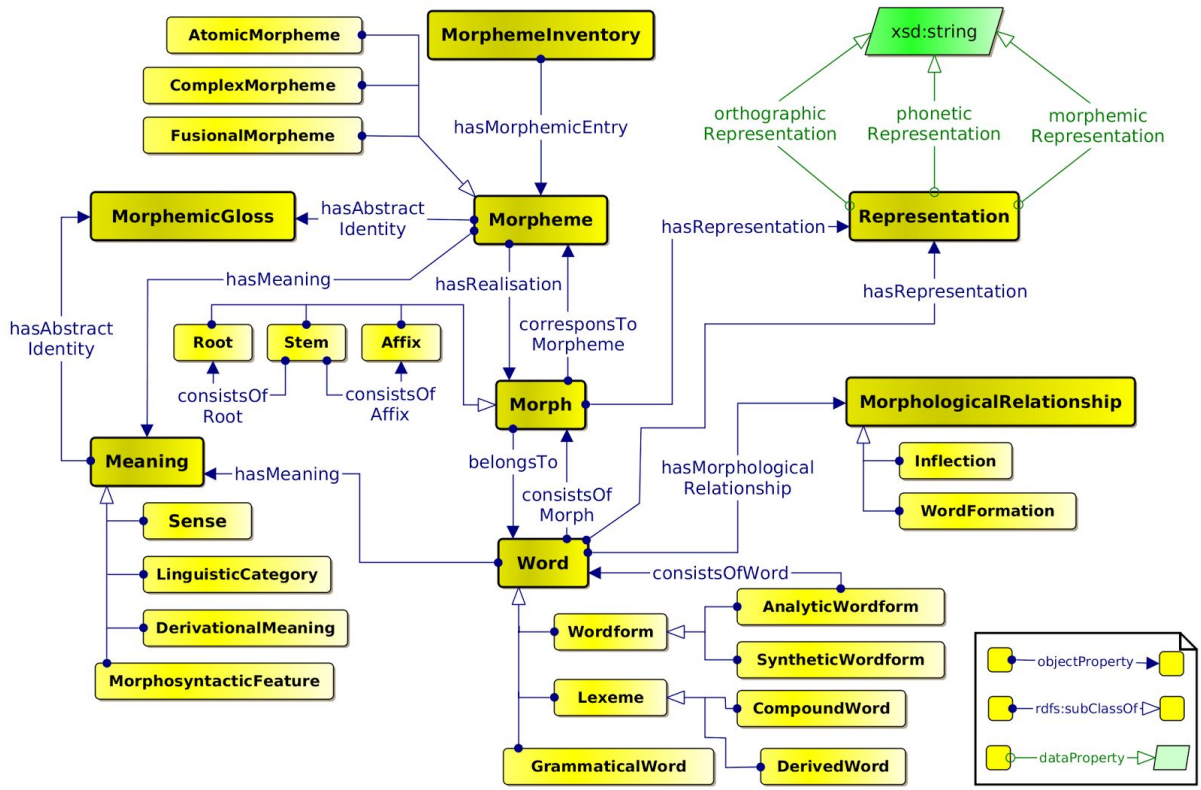

Abb. 2: Übersicht über das MMoOn-Core-Modell
(Quelle: http://mmoon.org/wp-content/uploads/2016/02/mmoon-core-v3.png)

Die Daten der Datenbank sind als Ontologie in RDF dargestellt, also in Form von Tripeln. Die Datenbank liegt uns in der Turtle-Serialisierung vor.
Das Datenmodell basiert auf der Modellierung der "**M**ultilingual **Mo**rpheme **On**tology" (MMoOn[5]) von Frau Bettina Klimek. Die entsprechenden Beziehungen können der Abbildung 2 entnommen werden. Aufbauend auf dieser grundlegenden Datenstrukturierung (core) existieren noch zwei "Schema"-Dateien und zwei "Inventory"-Dateien, jeweils für die Sprachen Deutsch und Hebräisch. Um die Entwicklung der Software und den Umgang mit den Daten zu erleichtern, werden Softwarekomponenten, die sich auf die Datenbank beziehen, mit den wesentlich weniger umfangreichen deutschen Dateien getestet.

[5] http://mmoon.org/