Design Dokument für TODO

Christian Stricker Alisha Klein David Klopp
Markus Vieth

7. Januar 2016

Inhaltsverzeichnis

Teil II.

LowLevelDesign

Einleitung

In diesem Dokument wird die Implementierung unseres Systems anhand verschiedener Klassen-, sowie Sequenzdiagramme dargestellt. Hierbei wird sowohl auf die Interaktion der verschiedenen Klassen untereinander, wie auch auf die hierfür verwendeten Pattern eingegangen.
Das System, sowie alle Angaben zum System, beziehen sich dabei auf das "Architectural Design Document for TODO“ vom 8. Dezember 2015.

1. Apache und Tomcat

1.1. Warum Apache verwenden?

Der Apache HTTP Server ist der meist verbreitete Webserver im Internet. Wir verwenden den Apache HTTP Server, da er quelloffen ist und aktiv weiterentwickelt wird. Durch diese Quelloffenheit ist der Server auf unterschiedlichen Plattformen lauffähig und leicht installier- und konfigurierbar.

1.2. Warum Tomcat verwenden?

Apache Tomcat ist open source und auf Grund seiner Java Implementierung ebenfalls, wie der Apache HTTP Server, plattformunabhängig. Da Tomcat die offizielle Referenzimplementierung für JavaServer Pages (JSP) und Servlets ist, ist dieser stets aktuell und wird dementsprechend aktiv weiterentwickelt. Die Implementierung der Servlets ist unabhängig von der Spezifikation des Webservers und sehr effizient, weshalb sie leicht portiert werden kann. Die Schnittstellen in Tomcat stellen eine klare Trennung zwischen Logik und Sicherheit bereit. Letztere wird durch die Verwendung der JavaVM weiter erhöht. All diese Punkte machen es leichter möglich ToDo effizient und sicher zu implementieren.

2. Präsentationsebene

Der Server implementiert verschiedene Servlets, die die Anfragen eines Clients entgegennehmen und beantworten. Hierzu werden die Anfragen des Nutzer ausgewertet und in der Klassenhierarchie an die zuständigen Instanzen weitergeleitet. Zu jedem REST Endpoint den das System bereitstellt existiert ein entsprechendes Servlet, welches von dem javax.servlet.http.HttpServlet abgeleitet ist. Jede Subklasse überschreit hierbei die ”doGet” und ”doPost” Methode, um die gewünschte Operation entsprechend der Art der Anfrage auszuführen. Hierzu kommuniziert die Servlet Klasse mit einer Subklasse des EntityControllers, der nach dem Model-View-Controller Pattern mit dem EntityModel und dem EntityView interagiert. Der EntityView realisiert hierbei den dynamischen Web-Inhalt, der dem Benutzer auf Anfrage angezeigt wird. Für jede verfügbare Ansicht existierten entsprechende Subklassen dieser drei MVC Klassen, die den jeweils benötigten Inhalt bereitstellen.

3. Logikebene

Jedes Package, Dataset, Model, jeder Algorithm und jede Prediction leiten von der Klasse Entity ab und haben somit einen eindeutige id, sowie uri und einen Namen. Über die Kommunikation mit den darüber liegenden Schichten wird der PluginLoader instruiert je nach Bedarf die jeweilig angeforderten Entities zu erzeugen bzw. abzurufen. Durch die Anbindung an die Weka-Library können die Datsets oder die Models verarbeitet werden, um so neue Predictions zu generieren. Sämtliche administrativen Anfragen die den User oder Usergoups betreffen werden durch den UserManager gehandhabt. Dieser kooperiert über den Adapter mit dem PrivilegeManager, dem PackageManager und dem SecurityManager um die Anfragen zu beantworten. Der Adapter implementiert hierbei das Singleton-Pattern und verwaltet die Instanzen der verschiedenen Manager. Durch diese Implementierung ist ein Zugriff auf sämtliche Kernkomponenten des Systems leicht von verschiedenen Klassen aus möglich. Der Datenzugriff auf die Datenbank erfolgt vollständig über die Klasse Dataaccess, die ebenfalls an den Adapter angebunden ist. Diese Klasse bildet hierbei einen Wrapper um die SPARQL Abfragesprache, um die Anfragen an die Datenbank zu formulieren.

4. Sequenzdiagramme