Design Document

Christian Stricker David Klopp Markus Vieth

2. Dezember 2015

Inhaltsverzeichnis

Teil I

Architectural Design

Kapitel 1

Einleitung

Im Folgenden werden in diesem Dokument verschiedene Perspektiven des zu entwickelnden Systems betrachtet. Dazu wird das System in Teilsysteme zerlegt und deren Verhalten aufgezeigt.

Das System, sowie alle Angaben zum System, beziehen sich dabei auf das „Requirements Document for TODO“ vom 20. November 2015.

Kapitel 2

Externe Sicht

Das System, als Web-Applikation, interagiert mit anderen Systemen in seiner Umgebung. TODO kommuniziert zur Übertragung von Daten mit mehreren Clients, welche Anfragen senden und Antworten empfangen, und mit weiteren Servern um Daten in den Datenbanken auszutauschen.

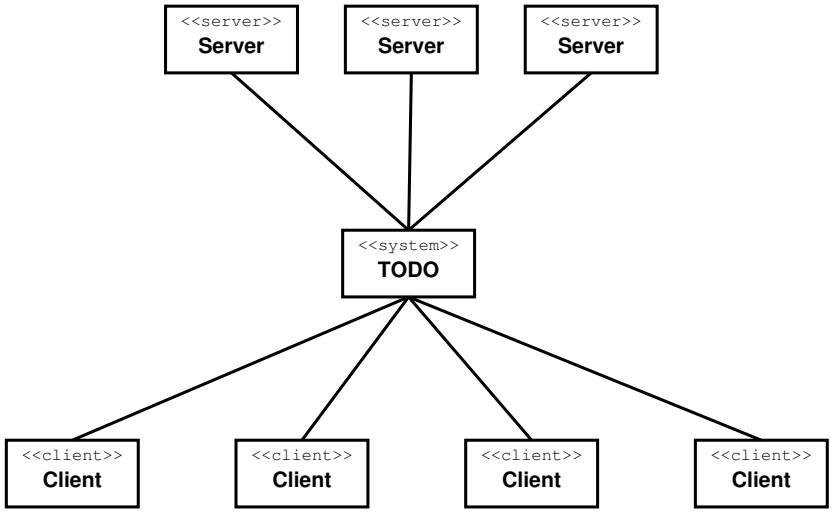

Abbildung 2.1: Context Diagram des Systems im Bezug zu seiner Umgebung

Die Kommunikation zwischen dem TODO und anderen Servern läuft dabei über das REST-Interface ab. Auch die Clients nutzen REST um Anfragen an das System zu stellen. Die Verbindung wird über HTTPS aufgebaut.

Kapitel 3

Interaktionssicht

Kapitel 4

Struktursicht

4.1 Microkernel

Das System stellt seine Funktionalität über die WEKA-Libary zur Verfügung. Damit diese arbeiten kann, werden Algorithmen benötigt. Um eine dynamische Ergänzung der Algorithmen zu ermöglichen wird WEKA als Mikrokernel implementiert. So können die Algorithmen als interne Server, wenn benötigt, geladen werden und neue Algorithmen können hinzugefügt werden, ohne das der WEKA-Quellcode bearbeitet werden muss. Alle Anfragen an WEKA laufen dabei über eine Datenschnittstelle, welche verschiedene Funktionalitäten für den Client bereitstellt. So kann die Datenschnittstelle zurückgeben, welche Algorithmen von einem bestimmten Datensatz unterstützt werden oder ob ein zu erstellendes Modell bereits in der Datenbank vorhanden ist. WEKA übernimmt die Berechnung eines Modells und die Auswertung eines Datensatzes (bzw. eines Algorithmus). Der Web-Server übernimmt die Rolle eines Adapters, welcher die REST-Anfragen des Clients auswertet und weitere Instruktionen einleitet.

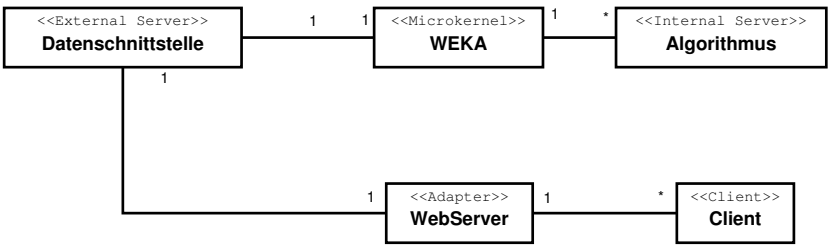

Abbildung 4.1: Microkernel-Pattern mit WEKA

4.2 Reflection

Um dynamisch später hinzugefügte Algorithmen auch in der Ein- und Ausgabe zu unterstützen, werden die Komponenten „PluginLoader“ und „Plugin“ verwendet. Der PluginLoader handhabt die Plugins und erstellt diese falls notwendig. Um dies zu tun, hat der PluginLoader Zugriff auf die Algorithmen. Somit kann der PluginLoader ein Plugin für die Eingabe von Parametern erzeugen, indem er sich von dem entsprechenden Algorithmus die Anzahl der benötigten Parameter ausgeben lässt. Analog kann der PluginLoader ein Plugin zur Ausgabe erzeugen, indem er sich zurückgeben lässt wie die Ausgabedaten aussehen und diese Anhand des gewünschten Ausgabeformats entweder als plain-text, html-text, JSON, PNG-Grafik oder SVG+XML-Grafik aufbereitet.

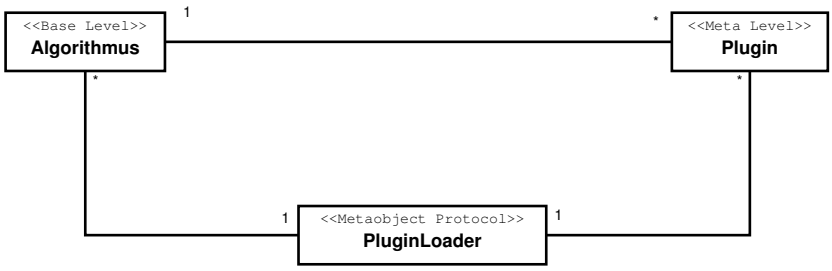

Abbildung 4.2: Reflectoin-Pattern mit PluginLoader

4.3 Layer

Um den Zugriff auf die Datenbank zu beschränken, wird zwischen die Datenschnittstelle und die Datenbank eine Benutzerverwaltung geschaltet. Diese überprüft nun bei jeder Anforderung auf einen Zugriff auf die Datenbank, ob dieser vom aktuellen Nutzer erlaubt ist oder nicht.

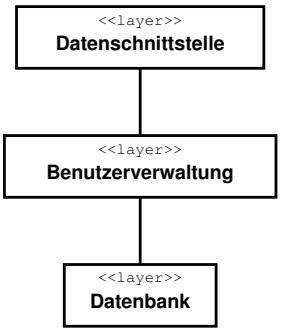

Abbildung 4.3: Layer-Pattern mit Benutzerverwaltung

Das Layer-Pattern hilft einen unerlaubten Zugriff auf die Datenbank zu verhindern und stellt sicher, dass jeder zugriff erst über die Benutzerverwaltung erlaubt wird. Ein weiterer Vorteil stellt die Flexibilität der einzelnen Schichten dar, so kann die Datenschnittstelle statt der lokalen Datenbank auch die Datenbank eines anderen Servers über REST abgerufen, da die Kommunikationswege innerhalb des Layer-Pattern identisch sind.