Design Dokument für TODO

Christian Stricker David Klopp Markus Vieth

3. 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.

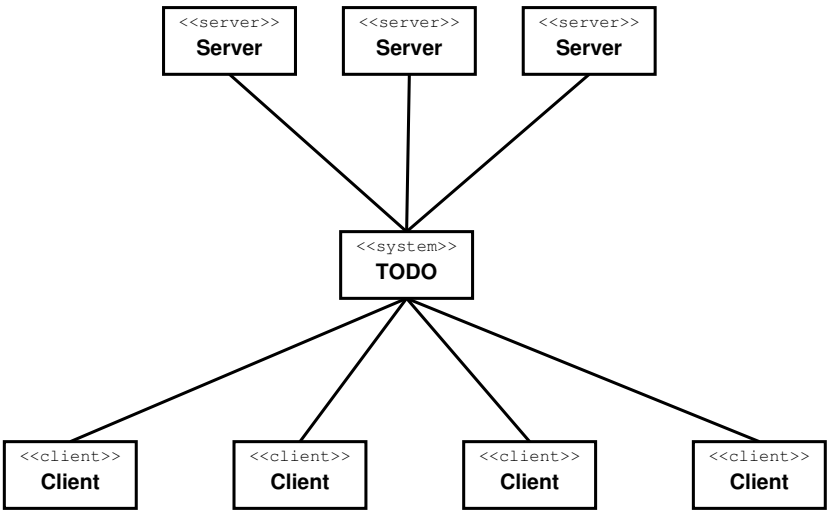

Abbildung 2.1: Context Diagram des Systems im Bezug zu seiner Umgebung

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. Der Nutzer kann zur Kommunikation mit dem System jegliche Software nutzen, welche den Austausch von Daten über HTTPS unterstützt, insbesondere Browser wie Firefox oder Google Chrome. Ein Administrator hat die Möglichkeit über einen Client oder direkt an dem Rechner, auf dem das System läuft, zu arbeiten.

Kapitel 3

Struktursicht

3.1 Genutzte architektur Pattern

3.1.1 Client-Server

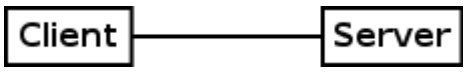

Abbildung 3.1: Client-Server-Modell

Client Der Client ist der für den Nutzer sichtbare Teil des Systems, also die Website, über die auf die serverseitigen Anwendungen zugegriffen werden kann.

Server Der Server bietet dem Nutzer den Dienst an Modelle anhand von schon vorhandenen oder von ihm hochgeladenen Datensätzen und Algorithmen zu erstellen. Hierzu verwaltet der Server eine Datenbank, die entsprechende Nutzerdaten, Modelle und Algorithmen beinhaltet.

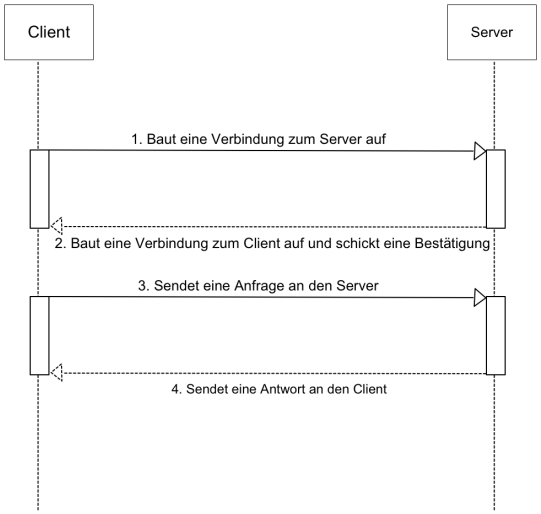

Abbildung 3.2: Client-Server-Sequenzdiagramm

Der Client versucht eine Verbindung zum Server aufzubauen. Der Server versucht ebenfalls bei Anfrage eine Verbindung zum Client aufzubauen und schickt diesem eine Bestätigung, dass die Kommunikation erfolgen kann. Danach kann der Client Datensätze und Algorithmen hochladen und dem Server eine Anfrage zum Modelle berechnen oder downloaden schicken.

Was spricht für das Client-Server-Modell?

Das Client-Server-Modell wird verwendet, wenn eine Datenbank oder ein Service von verschiedenen Orten her abrufbar sein soll. Für unser System sind beide dieser Aspekte gegeben. Verschiedene Nutzer können so als Client auf das System zugreifen und Daten hoch- und runterladen. Änderungen am System können rein serverseitig erfolgen ohne Zutun oder Benachrichtigung des Clients, was zu einer erhöhten Flexibilität und Stabilität führt.
Da das System über das REST-Protokoll mehrere Server adressieren können soll, ist es mehr als sinnvoll das Client-Server-Pattern zu verwenden, da andernfalls eine Implementierung dieses Protokolls nur schwer realisierbar wäre.

3.1.2 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.

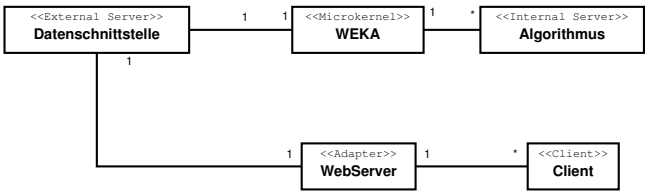

Abbildung 3.3: Microkernel-Pattern mit WEKA

Was spricht für den Microkernel?

Das System soll in der Lage sein stetig mit neuen Algorithmen versorgt zu werden. Das Microkernel-Pattern erlaubt dies, ohne im späteren Verlauf oder gar für jeden Algorithmus den Quellcode verändern zu müssen. Dies erlaubt eine große Flexibilität hinsichtlich der Möglichkeit verschiedene Algorithmen zu verwenden. Ein weiterer Aspekt ist die Trennung von WEKA und den Algorithmen. Dies verringert das negative Beeinflussen beider Parteien untereinander. Des Weiteren erlaubt es ein einfacheres ändern (z.B. Updaten, erweitern der Funktionalität) der Komponenten. In Anbetracht der Vorteile, sind die Performance-Einbußen durch das Pattern nicht weiter relevant.

3.1.3 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.

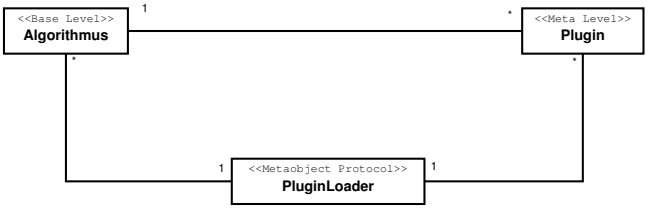

Abbildung 3.4: Reflectoin-Pattern mit PluginLoader

Was spricht für Reflection?

Da das System ständig mit Algorithmen erweiterbar sein soll, wäre es hinderlich, wenn jede Eingabe- und Ausgabemaske für die Algorithmen extra geschrieben werden müssten. Deshalb ist es sinnvoll mithilfe des PluginLoaders diese Masken dynamisch mithilfe der Eingenschaften der Algorithmen zu erzeugen und als Plugins zu speichern. Soll nun das Design der Webseite angepasst werden, eine neue Ausgabe möglich sein oder eine andere Sprache unterstützt werden, so reicht eine Anpassung des PluginLoaders, um passende neue Plugins zu erstellen. Die Permance-Verluste sind an dieser Stelle akzeptabel, da keine Umfangreichen Operationen durchgeführt werden müssen, um den Prozess auszuführen. Auch eine ungewollte Manipulation ist auszuschließen, da der Zugriff auf die Komponenten des System durch den Nutzer sehr stark limitiert ist.

3.1.4 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.

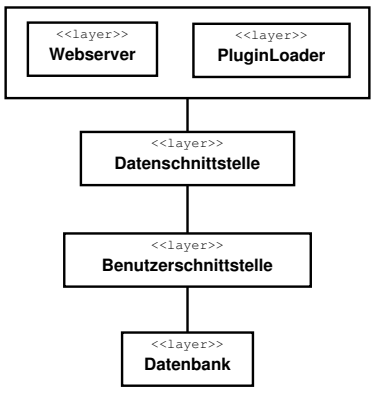

Abbildung 3.5: Layer-Pattern mit Benutzerverwaltung

Was spricht für das Layer-Pattern

Das Layer-Pattern hilft einen unerlaubten Zugriff auf die Datenbank zu verhindern und stellt sicher, dass jeder zugriff erst über die Benutzerschnittstelle 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 abrufen, da die Kommunikationswege innerhalb des Layer-Pattern identisch sind.

3.1.5 Model-View-Controller

Um bestimmte Darstellungskomponenten, im weiteren als Plugins bezeichnet, in das UI (in unserem Fall die Website) einzubinden, wird das MVC-Pattern verwendet. Solche Plugins können z.B. die Ausgabe von Ergebnissen als Text oder Grafik sein, aber auch Eingabemasken zum Erstellen von Modellen.

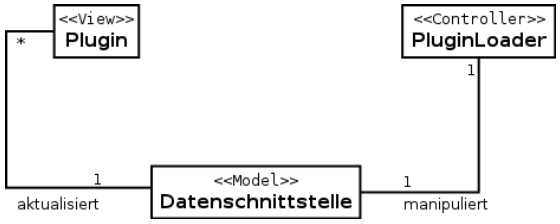

Abbildung 3.6: MVC-Pattern zum Website anzeigen

Sollte der Benutzer z.B. ein Modell generieren wollen oder sich ein Ergebnis anzeigen lassen, so formuliert der PluginLoader eine Anfrage an die Datenschnittstelle, um mögliche Änderungen an diese zu senden. Nach diesem Aktualisierungsprozess wird ein Plugin durch den PluginLoader generiert, das über die Datenschnittstelle die notwendigen Informationen bezieht. Sollten hierbei Anfragen an andere Server nötig sein, so werden diese ebenfalls von der Datenschnittstelle mittels des REST-Protokolls durchgeführt. Das fertige Plugin wird nun in das Haupt-UI integriert.

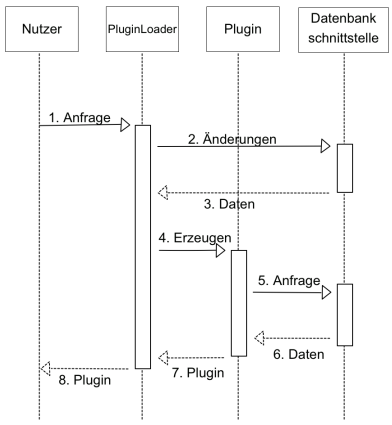

Abbildung 3.7: MVC-Pattern Sequenz zum Generieren der Website

Was spricht für Model-View-Controller?

Das MVC-Pattern ermöglicht es, das System leicht zu erweitern. Durch die Einbindung verschiedener Plugins in das System wird gewährleistet, dass auf neue Gegebenheiten schnell reagiert werden kann. Sollte z.B. ein Algorithmus eine spezifische Ausgabe benötigen, könnte speziell für diesen ein neues Plugin erstellt werden. Des Weiteren erleichtert diese Aufteilung die Wartung des Systems enorm, da einzelne Plugins für sich getestet werden können, ohne in das Gesamtsystem eingebettet zu sein.

3.2 Nicht genutzte architektur Pattern

3.2.1 Presentation-Abstraction-Control

Was spricht gegen Presentation-Abstraction-Control?

Bei klar definierten Benutzer-System Interaktionen ist eine verschachtelte Struktur mittels Agenten, wie sie PAC vorsieht, zu umfangreich und nicht notwendig. Die Stärke von PAC liegt darin, verschiedene, bestehende Teilsystem miteinander zu verknüpfen. In unserem System ließe sich dies beispielsweise auf die Datenbank und die Implementierung von WEKA anwenden. Da allerdings nur diese beiden Komponenten Daten generieren können, ist es einfacher, diese über eine übergeordnete Komponente anzusprechen, als die hohe Komplexität des Pattern in Kauf zu nehmen. Der Fokus unsere Implementierung des Systems liegt auf der Erweiterbarkeit des UIs, die durch MVC leichter und effektiver gegeben ist.

3.2.2 Pipe and Filter

Was spricht gegen Pipe-and-Filter-Pattern?

Der große Vorteil des Pipe-and-Filter-Pattern besteht darin kontinuierliche Datenströme, mit Hilfe verschiedener Filterbausteine, asynchron und meist unabhängig von einander zu verarbeiten. Da das System allerdings hauptsächlich auf einfachen Dateiaustausch und kurze Anfragen an den Server basiert, reicht es vollkommen aus eine normale https Verbindung zu verwenden ohne eigene Filterströme zwischenzuschalten. Dieses Pattern wäre für die Kommunikation zwischen Client und Server zu mächtig und würde lediglich zu unnötigen Leistungseinbußen führen.

3.2.3 Blackboard

Was spricht gegen Blackboard?

Das hauptsächliche Einsatzgebiet des Blackboard-Patterns besteht darin, komplexe Sachverhalte auf kleinere Teilprobleme zu reduzieren und diese von sogenannten Experten lösen zu lassen. Anschließend wird das Gesamtergebnis aus den einzelnen Teilergebnissen zusammengesetzt. Unser System hingegen hat einen klar definierten sequenziellen Programmablauf:
- Benutzereingabe
- Auswertung der Eingabe
- Datenbankzugriff oder Berechnung.

Es macht daher keinen Sinn, Teilprobleme bilden zu wollen. Bei der spezifischen Implementierung eines Algorithmus könnte dieses Pattern eventuell Anwendung finden.

3.2.4 Broker

Was spricht gegen Broker-Modell?

Das Broker-Modell vermindert die Performance, da alle Komponente des Systems nur indirekt angesprochen werden. Des Weiteren hängt die Kommunikation der einzelnen Systeme von vielen Komponenten ab und ist deshalb Fehleranfällig. Sinnvoll wäre der Broker nur, wenn viele verschiedene Clients auf viele verschiedenen Server auf viele verschiedene Service zugreifen wollen. Die Anzahl der Services unseres Servers ist sehr überschaubar und deshalb ist der Broker zu ineffizient für unseren Server.

Kapitel 4

Interaktionssicht

4.1 Szenarien

4.1.1 Login

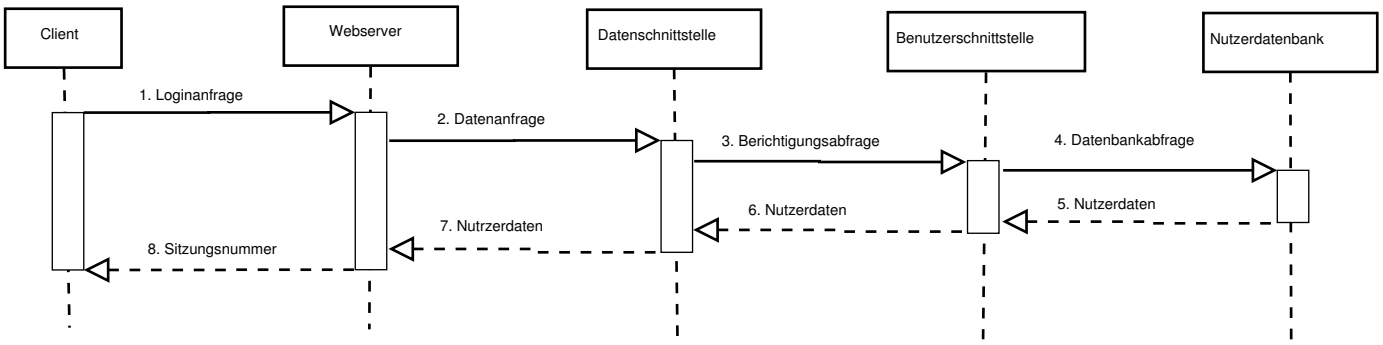

Abbildung 4.1: Anmeldung eines Benutzers

Nachdem der Benutzer die Startseite aufgerufen hat, kann er sich über ein Formular auf der Seite anmelden. Nach dem Klick auf die entsprechende Schaltfläche wird eine Login Anfrage an den Webserver geschickt. Dieser beauftragt die Datenschnittstelle die erforderlichen Nutzerdaten abzufragen. Hierzu wird die Benutzerschnittstelle angesprochen, die prüft, ob es sich bei den Anmeldedaten um einen registrierten und zugangsberechtigten Nutzer handelt. Sollte dies der Fall sein, so werden die angeforderten Daten aus der Nutzerdatenbank ausgelesen und in der selben Hierarchie bis zum Webserver nach oben gereicht. Dort angelangt wird dem Nutzer eine Sitzungsnummer zugewiesen und ihm zu Zugang zu seinem Account gewährt.

4.1.2 Algorithmus hinzufügen

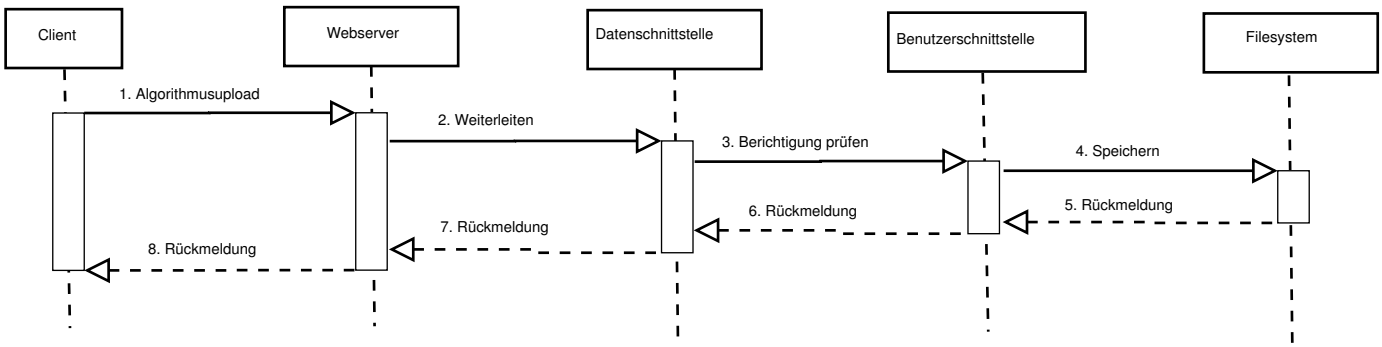

Abbildung 4.2: Upload eines Algorithmus

Der Nutzer schickt eine Anfrage zum Upload eines Algorithmus an den Webserver. Dieser delegiert die Nachfrage an die Datenschnittstelle, die diese ihrerseits an die Benutzerschnittstelle zustellt. Dort wird geprüft ob der Nutzer eine Berechtigung zum Upload eines Algorithmus hat, sprich ob er auf dem System registriert ist oder nicht. Sollte er die erforderlichen Kriterien erfüllen, so wird dem Admin der Algorithmus per E-Mail zugestellt und der Benutzer wird über die erfolgreiche Zustellung informiert. Nach Prüfung der übermittelten Daten kann der Admin diese in das Filesystem des Systems integrieren.

4.1.3 Modell aus vorhandenen Dastensatz berechnen

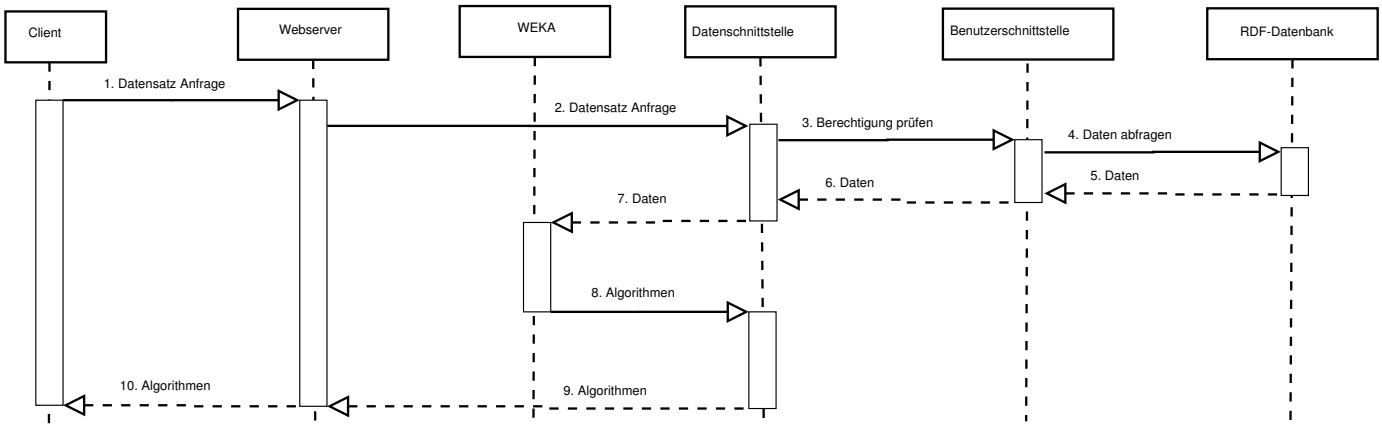

Abbildung 4.3: Auswahl eines Datensatzes

Um aus einem vorher hochgeladenen Datensatz ein Modell zu berechnen, muss der Nutzer diesen zuerst auswählen. Diese Aktion wird an den Webserver weiter-

geleitet, welcher wiederum eine Anfrage an die Datenschnittstelle sendet. Diese entnimmt den Daten der Anfrage wo sich der Datensatz befindet und sendet daraufhin eine Anfrage an die Benutzerschnittstelle, welche überprüft ob ein Zugriff auf die Daten durch den Nutzer erlaubt ist. Steht dem Zugriff auf die Daten keine Beschränkung entgegen, so leitet die Benutzerschnittstelle die Anfrage an die entsprechende Datenbank weiter. Diese gibt die angeforderten Daten über die Benutzerschnittstelle zurück an die Datenschnittstelle. Dort wird der Datensatz zur Analyse an das WEKA-Modul weitergeleitet, welches feststellt, welche der vorhandenen Algorithmen den Datensatz verarbeiten können. Diese Liste an Algorithmen wird wieder an die Datenschnittstelle zurückgegeben, welche wiederum die Daten an den Webserver weitergibt. Dieser gibt die Daten passend an den Client weiter, so dass diesem angezeigt werden kann, welche Algorithmen mit dem gewählten Datensatz verwendet werden können.

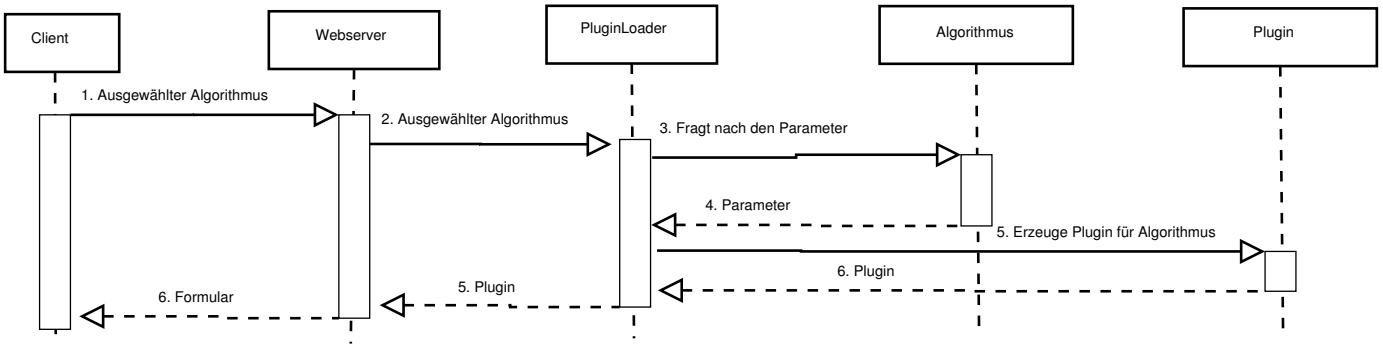

Abbildung 4.4: Auswahl eines Algorithmus

Hat der Nutzer nun einen Algorithmus ausgewählt, so wird dieser an den Webserver weiter gegeben. Der fordert nun den PluginLoader auf ihm ein Plugin zur Erzeugung eines Formulars für diesen Algorithmus zu geben. Dafür fragt der PluginLoader den Algorithmus, nach den Parametern, die er zur Ausführung benötigt. Aus dieser Information erzeugt der PluginLoader nun ein Plugin für diesen Algorithmus, welcher dann wiederum über den PluginLoader an den Webserver weitergegeben wird. Dieser erzeugt nun mit der Hilfe des Plugins ein Formular mit den Parametern, welches er wiederum dem Client schickt.

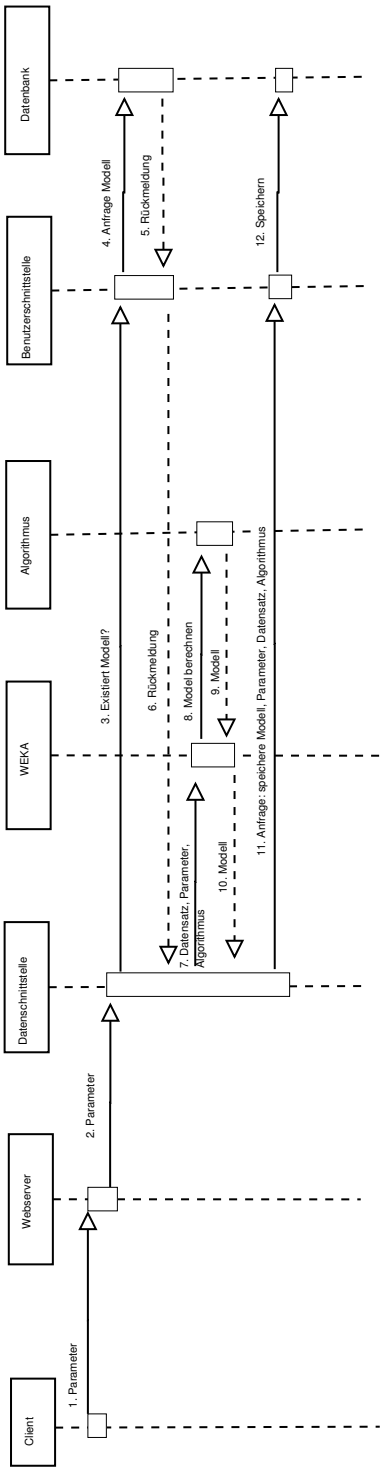

Abbildung 4.5: Berechnung eines Modells

Wurden schließlich Datensatz, Algorithmus und Parameter gesetzt, werden diese über den Webserver an die Datenschnittstelle übergeben. Diese überprüft nun ob ein Modell für diese Kombination an Datensatz, Algorithmus und Parametern in der Datenbank existieren, die Benutzerschnittstelle reglementiert dabei, welche Bereiche der Datenbank der Nutzer in seiner Rolle einsehen kann. Ob ein solches Modell existiert wird anschließend an die Datenschnittstelle weitergegeben. Existiert kein solchen Modell oder entscheidet sich der Nutzer dafür ein neues Modell zu erstellen, so übergibt die Datenschnittstelle die erforderlichen Daten an das WEKA-Modul. Dieses lässt nun von dem passenden Algorithmus das Modell berechnen und gibt dieses an die Datenschnittstelle zurück. Diese speichert nun das Modell mit den genutzten Daten in der Datenbank, da der Nutzer während der Berechnung nicht blockiert sein soll. Dieser kann anschließend zu einem beliebigen Zeitpunkt das Modell aus der Datenbank abrufen.

Kapitel 5

Anhang

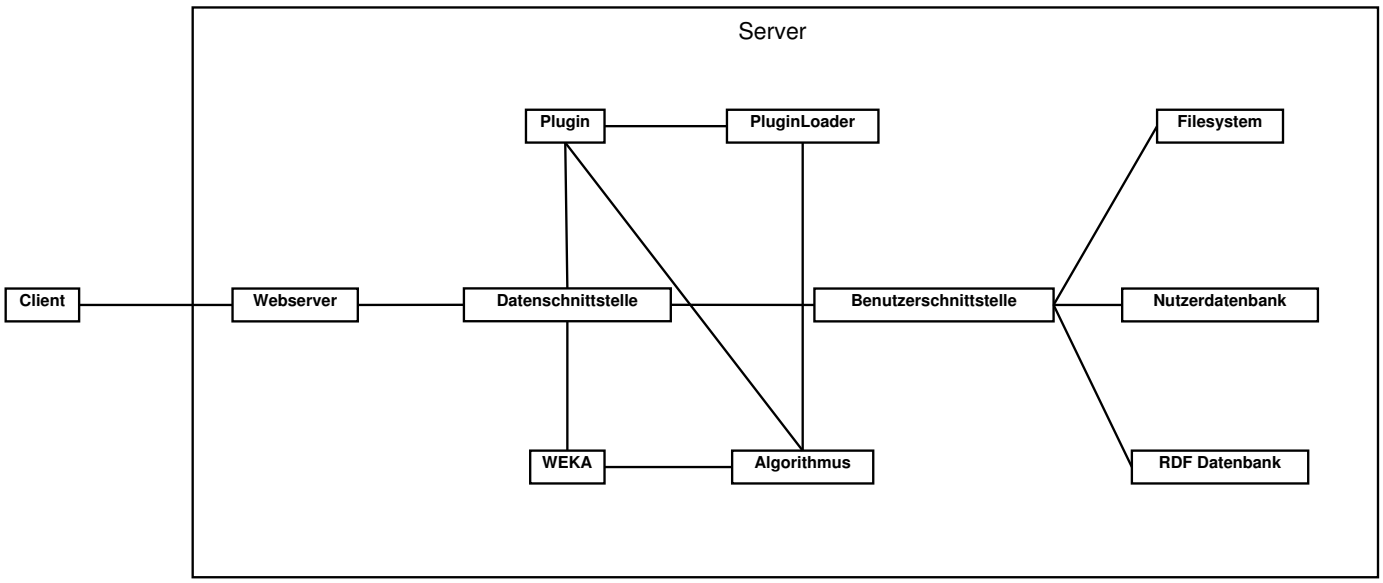

Abbildung 5.1: Skizze der Verhältnisse im gesamten System