Architekturdokument

AtmoCalc

Nicole Hauck Hanna Heinemann

Andreas Luft Lars Porth

SE 2014/2015

Einleitung

Dieses Architekturdokument beschreibt das zu entwickelnde System „AtmoCalc“ aus verschiedenen Perspektiven und auf verschiedenen Detailebenen. Alle Referenzen auf Anforderungen beziehen sich auf das gegebene Anforderungsdokument vom 09.12.2014[1]. Die Referenzen beziehen sich lediglich auf die Systemanforderungen des Dokuments.

Externe Sicht

Um dem Nutzer spezielle Informationen über die analysierten Moleküle bereitzustellen, greift „AtmoCalc“ auf externe Systeme über deren API zu (betreffend FR065). Hierbei werden die externen Systeme „ChemSpider“, „ToxBank“ und „MassBank“ angesprochen, diese externen Systeme benötigen keinerlei Kenntnis über den Aufbau der AtmoCalc-Anwendung. Diese externe Sicht wird in Abbildung 1 durch ein Context-Diagramm dargestellt, wobei anzumerken ist, dass ein Context-Diagramm nicht die Art der Verbindung zeigt, sondern nur, dass eine Verbindung besteht.

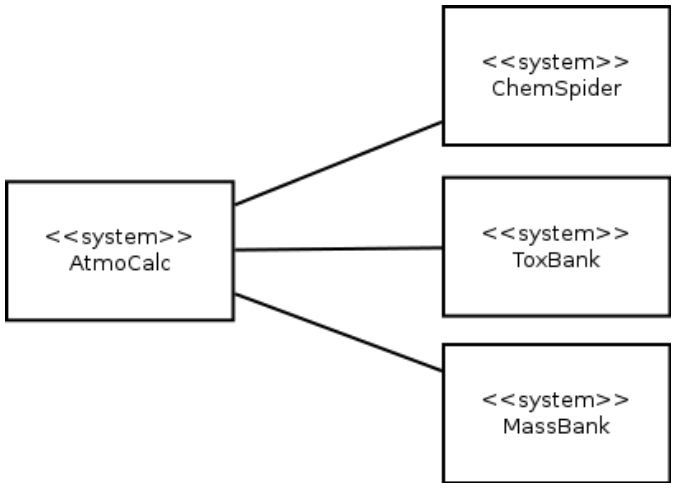


Abbildung 1: Context-Diagramm im Bezug auf andere Systeme

[1] https://reader.uni-mainz.de/WiSe2014-15/08-079-020-00/Lists/DocumentLib/main.pdf

Struktursicht

Allgemein

Betrachtet man das System als Ganzes zur Laufzeit, so erinnert dies stark an das Client-Server-Pattern. Bei diesem Pattern werden vom Server verschiedene Services bereitgestellt, in unserem Fall werden diese Services hauptsächlich durch die verschiedenen Algorithmen beschrieben. Es sollen gleichzeitig mehrere Algorithmen zur Analyse auf Daten angewandt werden können, wobei die nötigen Berechnungen auf der Serverseite stattfinden (vgl. NFR021). Auf die Daten müssen die Nutzer von überall auf der Welt zugreifen können (sofern sie die nötigen Rechte haben). Außerdem sollen weitere Algorithmen durch ein API genutzt werden können. Daher nutzen wir hier das Client-Server-Pattern, wobei die Web-Applikation den Client und die Datenbank den Server darstellt (vgl. Abbildung 2).

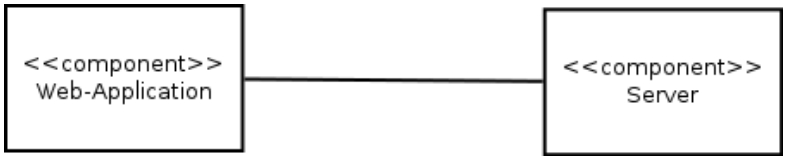


Abbildung 2: Die Web-Applikation als Client, zusammen mit dem Server

Client

Um den Aufbau des Clients darzustellen, verwenden wir das Layer-Pattern. Wir teilen den Client in drei Schichten: Präsentations-, Logik- und Datenebene (vgl. Abbildung 3). In der obersten Schicht, die der Präsentation dient, finden sich drei Komponenten für die verschiedenen Perspektiven: User Area (vgl. FR012-FR020), Visualisation (vgl. FR052-FR060, NFR021) zum Visualisieren der Daten, und Export/Import (vgl. FR057, FR061, FR027), hier kann der User Daten hochladen oder ein Dateiformat zum Download wählen. Die Möglichkeit des Wechselns zwischen den verschiedenen Perspektiven wird in Abbildung 3 durch die Verbindungen zwischen den Perspektiven dargestellt.

Die Aufgabe der Logikschicht besteht darin, den Programmablauf zu steuern. Sie enthält hauptsächlich die Logiken hinter den Perspektiven der Präsentationsebene und ist darüber auch mit dieser verbunden.

Die einzige Komponente des Clients in der Datenebene ist eine Komponente um mit dem Server zu kommunizieren, durch diese wird die Anforderung aus NFR021 realisiert, dass die Berechnungen auf Serverseite stattfinden sollen. Diese Kommunikationskomponente stellt gleichzeitig die einzige Verbindung zwischen Logik- und Datenebene dar.

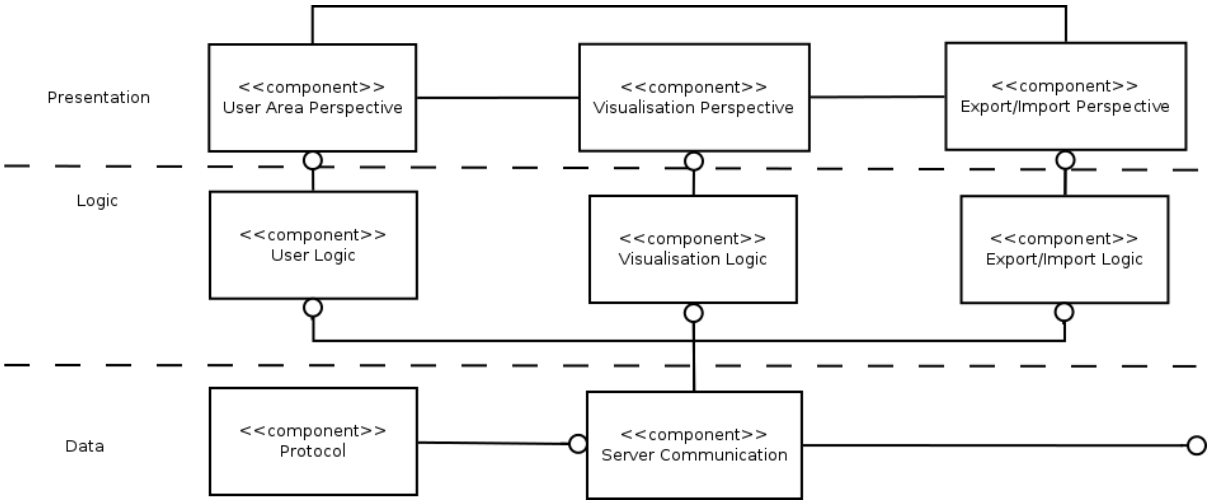


Abbildung 3: Grobübersicht des Clients

Wir nutzen hier das Layer-Pattern, da wir so bei Bedarf einzelne Schichten komplett ersetzen können, solange die Interfaces eingehalten werden. Dadurch lässt sich auch die in FR002-FR005 geforderte Variabilität durch Erweiterungen umsetzen. Die Kommunikation im Layer-Pattern erfolgt lediglich nach unten, dadurch bleiben die Interfaces stabil, und da alles in den unteren Schichten nur durch Anfrage aus der Präsentationsschicht geschieht, stellt diese Eigenschaft des Layer-Patterns keinerlei Probleme dar.

Im Folgenden betrachten wir die oberen beiden Schichten etwas ausführlicher. Als erstes sehen wir uns die „User Area“-Perspektive und die zugehörige Logik genauer an (vgl. Abbildung 4). Diese Perspektive lässt sich wiederum aufteilen in zwei Sichten. Zunächst gibt es die Registrierungsansicht, um sich am System anzumelden (vgl. FR010). Danach erhält man eine Verwaltungsansicht, über die dann eine Reihe unterschiedlicher Funktionen zur Verfügung stehen (vgl. FR013-FR019). Zum Beispiel kann man hier Datensätze verändern, für andere freischalten oder löschen. Welche Funktionen genau zur Verfügung stehen hängt vor allem von den Rechten ab, die der jeweilige Benutzer inne hat (vgl. FR022).
Um diese beiden Sichten darzustellen, verwenden wir das Model-View-Controller-Pattern. Dieses Pattern zerlegt interaktive Programme in drei Teile: Model, View und Controller. In unserem Fall bedeutet das zum Beispiel für den Login-Vorgang folgendes: Der Benutzer gibt seine Daten ein, dies wird durch den Login-View registriert, welcher daraufhin dem LoginController Bescheid gibt. In der Logikschicht befindet sich das LoginModel, das die Daten überprüft und dann dem LoginController eine Rückmeldung gibt, ob der Login erfolgreich war oder nicht. Zuvor übergibt das LoginModel dem CryptoModel die eingegebenen Daten, welches diese dann verschlüsselt. Der LoginController wandelt diese Information wiederum in eine Ausgabe für den Benutzer um, die diesem dann über den LoginView angezeigt wird.

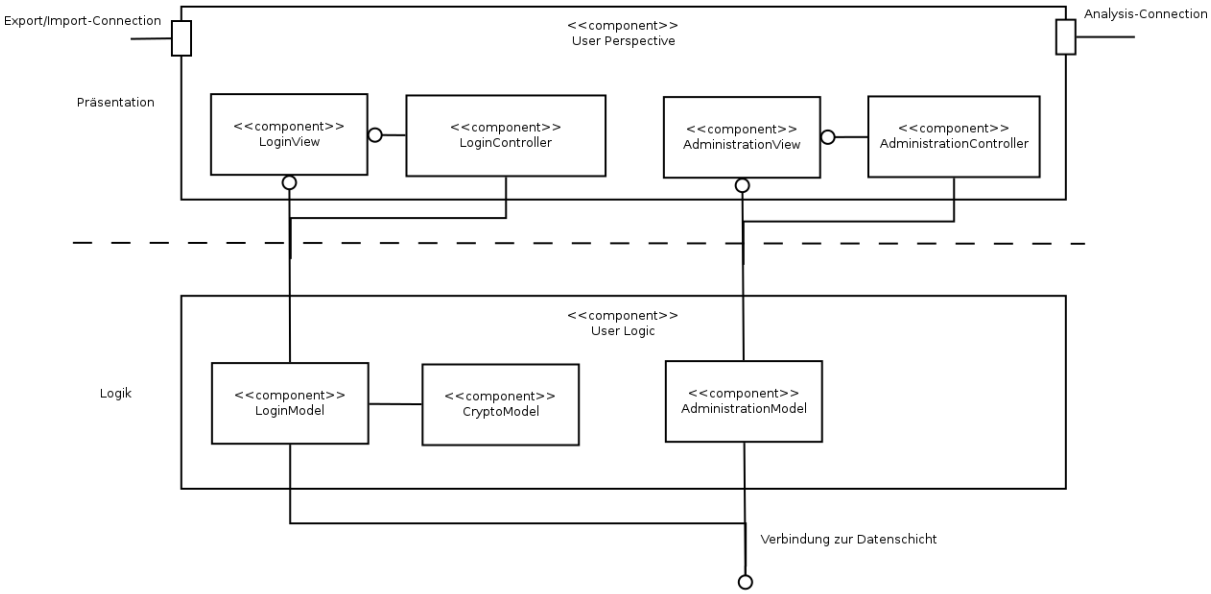


Abbildung 4: Die User-Perspektive im Detail

Über eine Schnittstelle ist die „User Area“-Komponente mit der „Visualisation“-Komponente verbunden, welche wir im Folgenden betrachten (vgl. Abbildung 5). Die einzelnen Komponenten der „Visualisation“-Perspektive lassen sich ebenfalls durch das Model-View-Controller-Pattern darstellen. Besonders nützlich ist dieses Pattern für die Visualisierungskomponente. Das Pattern erlaubt für ein Model und einen Controller mehrere Views. Dadurch können auf gleiche Weise analysierte Daten in unterschiedlichen Formen angezeigt werden (zum Beispiel Balkendiagramm, Graph etc.).

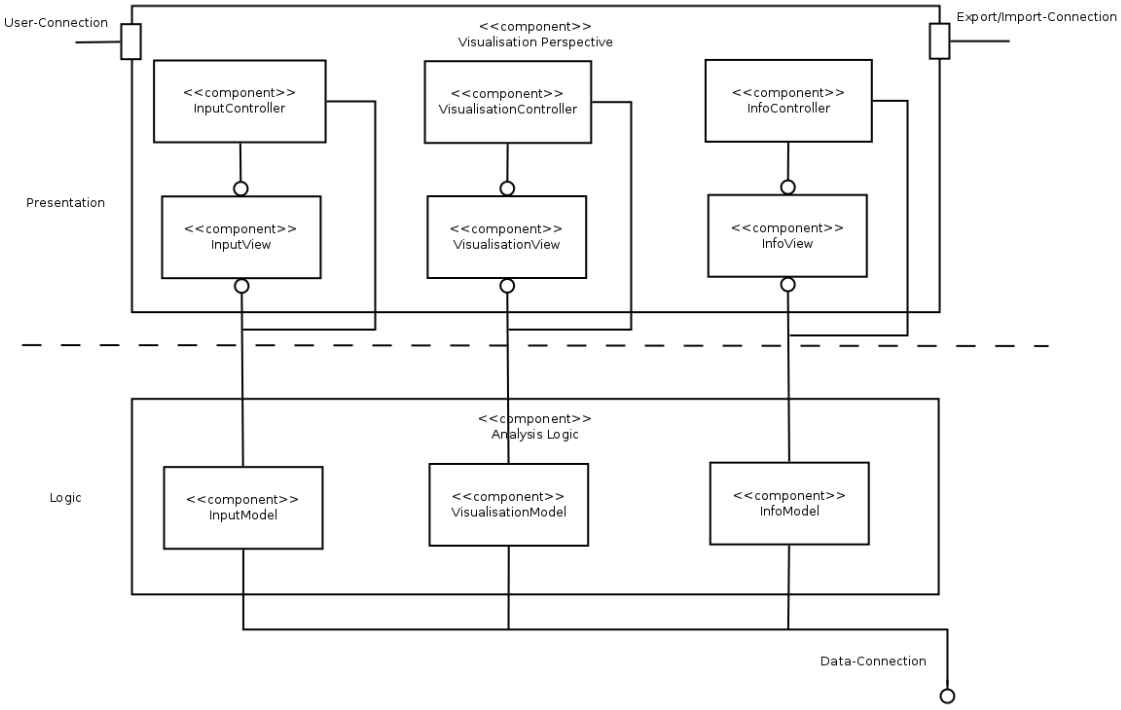


Abbildung 5: Die Visualisation-Perspektive im Detail

Sowohl von der „User Area“-Komponente als auch von der „Visualisation“-Komponente gibt es eine Verbindung zur „Export/Import“-Komponente (vgl. Abbildung 6). Auch diese lässt sich wieder zerteilen, nämlich in Export und Import. Auch diese beiden Teile lassen sich wieder durch das Model-View-Controller-Pattern wie oben beschrieben darstellen. Beim Export gibt es zusätzlich noch ein „Converter Module“, welches vom Server umgewandelte Daten zum Download im PDF-Format (vgl. FR061) bzw. Graphen in einem ausgewählten Bildformat (JPG oder PNG, vgl. FR058) anbieten.

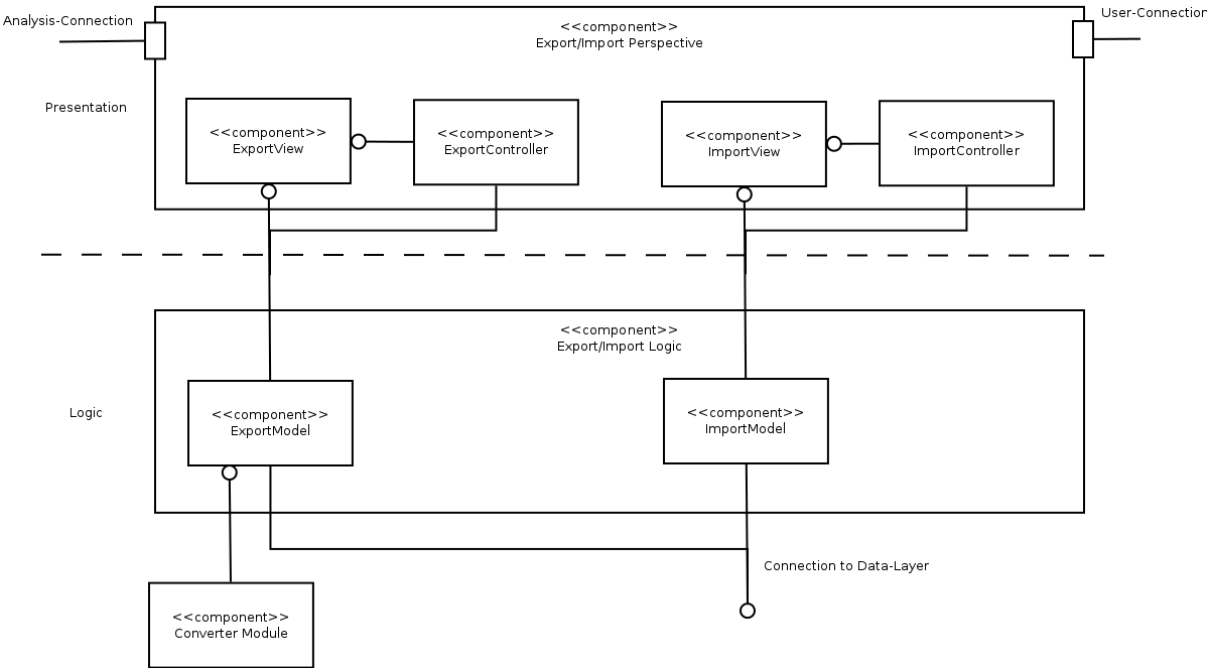


Abbildung 6: Die Export/Import-Perspektive im Detail

Server

Auch bei der Darstellung des Servers nutzen wir das Layer-Pattern (vgl. Abbildung 7). Auf der obersten Ebene existiert die Webbrowserschnittstelle, mit der der Server mit dem Client kommunizieren kann. Darunter befindet sich die Login-Komponente und die Formularverwaltung, welche sich um die Anmeldung und Registrierung sowie die Eingaben des Nutzers im System kümmern. Danach folgt die Rechteverwaltung, die für das korrekte Zuordnen von Rechten auf bestimmte Analysen von Nutzern zuständig ist. Diese Analysen - und deren (exportierte) Visualisierungen - werden auf der nächsten Schicht verwaltet. Die unterste Schicht ist die Datenbank, in der alle Daten, besonders im Bezug auf Datensätze, Nutzer und Analysen, gespeichert werden.
Die Vorteile der Verwendung dieses Patterns ergeben sich durch die bereits oben genannten Punkte.

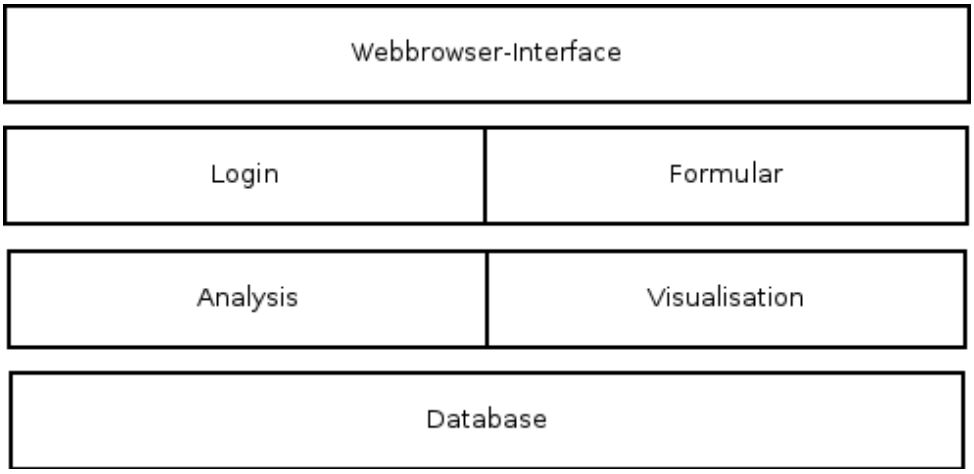


Abbildung 7: Grobe Layer-Übersicht des Servers

Diese Abbildung lässt sich näher spezifizieren (vgl. Abbildung 8): Damit der Server auch von außen verwaltet werden kann, wird die „Admin User Settings“- und die dazugehörigen Settings-Komponente benötigt (vgl. NFR003). Der Datenaustausch zwischen Client und Server wird durch die Kommunikationskomponente realisiert, welche die übertragenen Pakete zunächst standardisiert. Da die Analyse der Datensätze auf dem Server realisiert wird, wird hierzu die Analysis-Logic-Komponente benötigt. Diese benötigt keine direkte Verbindung mit der Datenbank, da Analysen die bestehenden Daten nicht verändern sollen (vgl. FR046). Um Informationen zu Molekülen

bereitzustellen wird die Info-Logic benutzt, die wiederum auf Datenebene mit dem External-Database-Provider per API auf die Datenbanken der externen Systeme zugreifen kann (vgl. Abbildung 1). Damit alle internen Komponenten mit der Datenbank kommunizieren können, wird auf der Datenebene zusätzlich noch die eigentliche Datenbank-Komponente mit der Database-Abstraction-Komponente benötigt, welche die Kommunikation soweit abstrahiert, dass sonstige Systeme unabhängig der verwendeten Datenbanktechnologie arbeiten können.
Zusätzlich soll die Möglichkeit bestehen, Algorithmen über eine API ins System einzubinden. Bei diesem Plug-in-Mechanismus bietet sich das Reflection Pattern an. Bei diesem wird das System in zwei Ebenen aufgeteilt, das Meta Level und das Basis Level. Für jede Komponente, die aktualisiert werden können soll, gibt es ein Meta Objekt. Dieses dient als Schnittstelle zu den Komponenten, weil jede Komponente von so einem Element abgeleitet wird. Es hat den Vorteil, dass das Hauptsystem zunächst kontrollieren kann, welche Daten den externen Algorithmen übergeben werden. Somit ist die Datenbank vor Missbrauch - beispielsweise versehentliches Löschen der Daten oder Auslesen von Nutzerdaten - geschützt. Außerdem muss der externe Algorithmus den Aufbau des Systems „hinter“ der API nicht kennen, wodurch die Entwicklung dieser einfacher wird. Die Standardalgorithmen werden dabei nicht verändert (vgl. FR009).

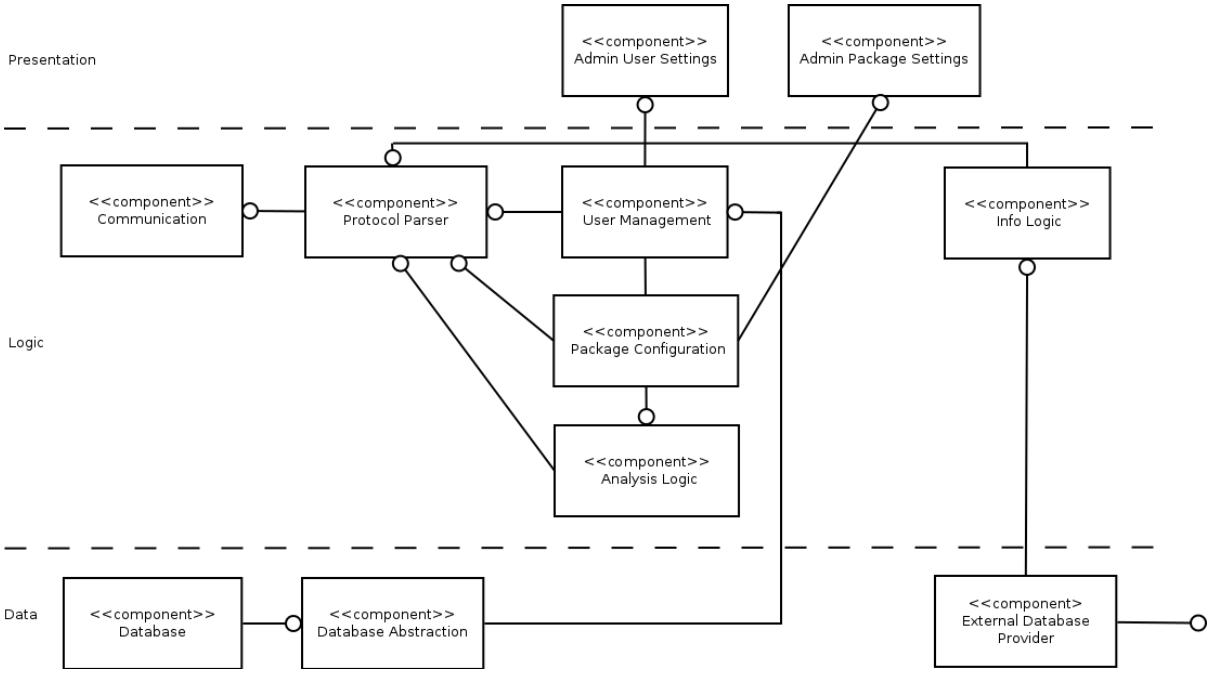


Abbildung 8: Der Server im Detail

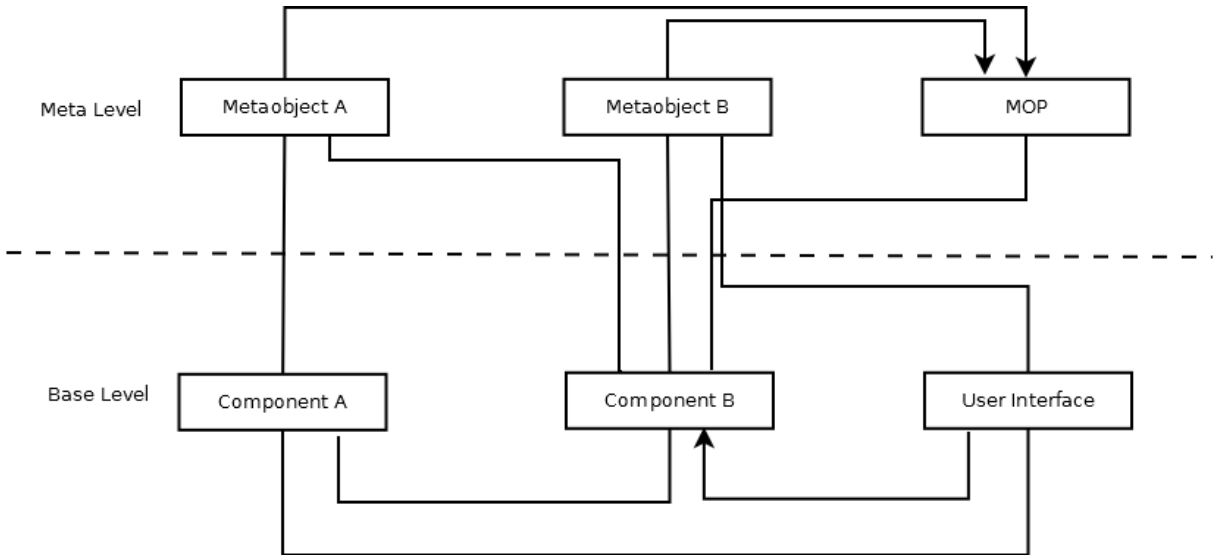


Abbildung 9: Reflection Pattern

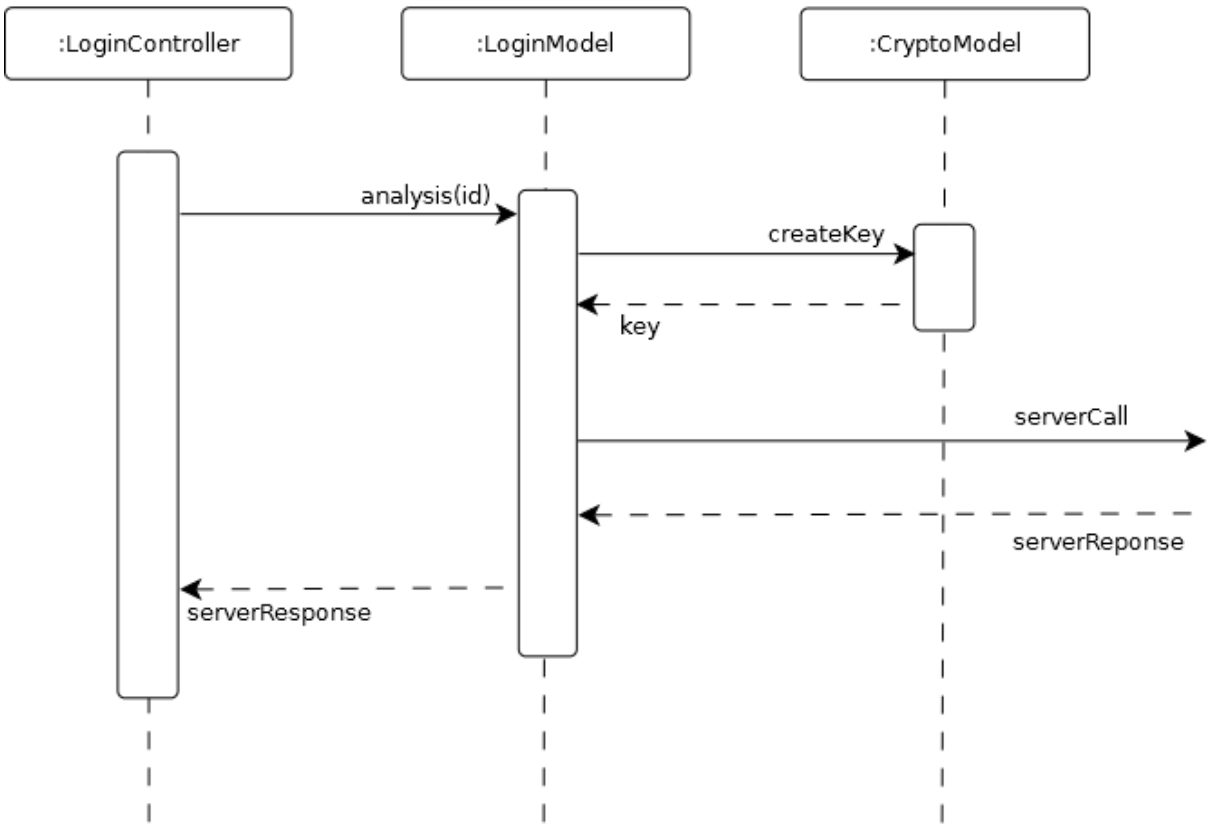
Interaktionssicht

Allgemein

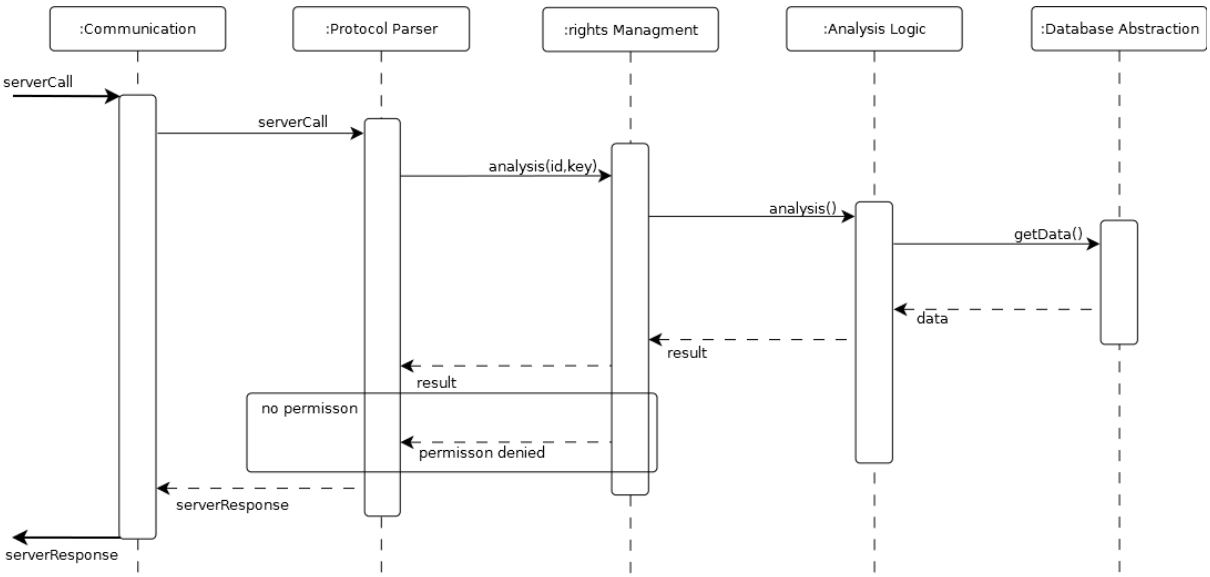
Im Folgenden wird die Interaktion zwischen den verschiedenen, bereits beschriebenen Komponenten des Systems in unterschiedlichen Szenarien dargestellt. Dabei steht die Visualisierung des Datenaustauschs mithilfe von Sequenz-Diagrammen im Vordergrund; eine exakte Definition der möglichen Methodenaufrufe soll dabei nicht vorgeschrieben werden.

Analysieren eines Datensatzes

Aus Client-Sicht:

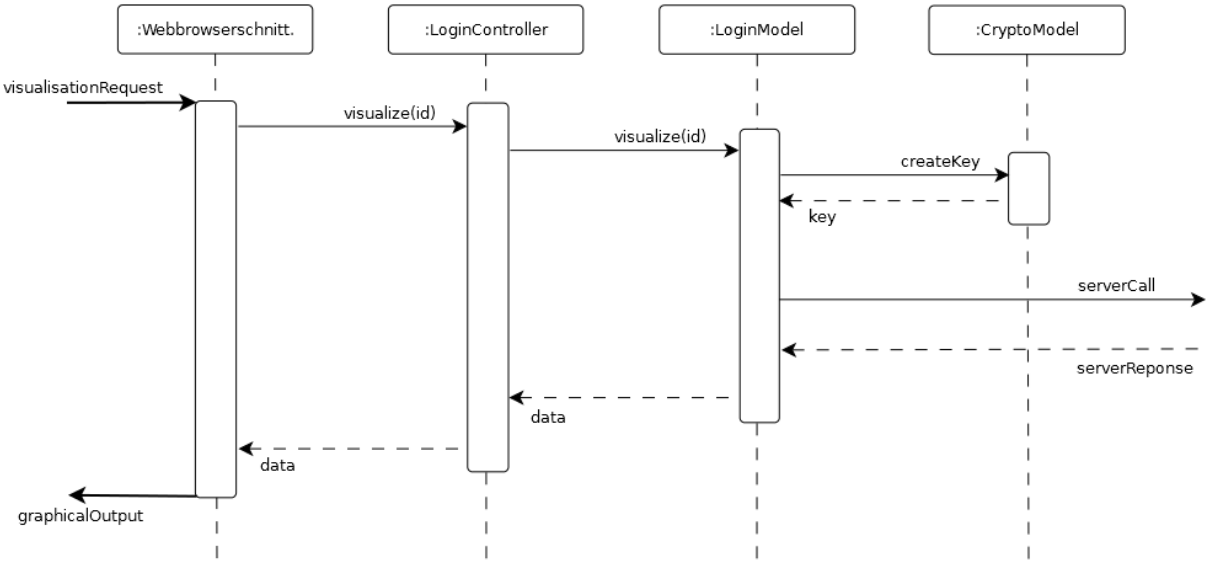


Aus Server-Sicht:

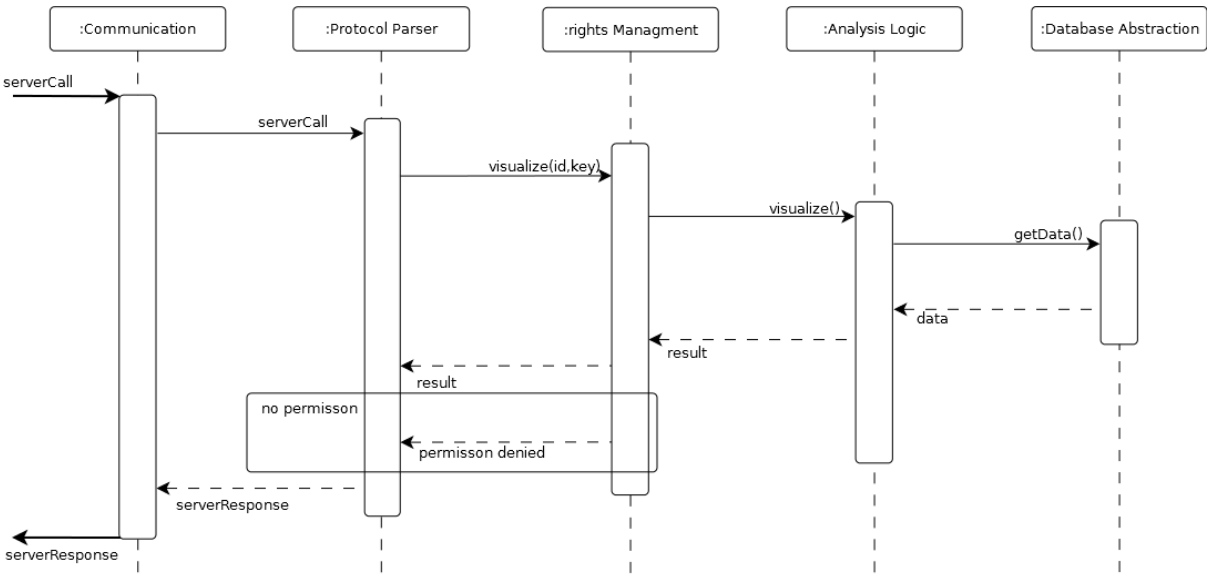


Visualisieren einer Analyse

Aus Client-Sicht:

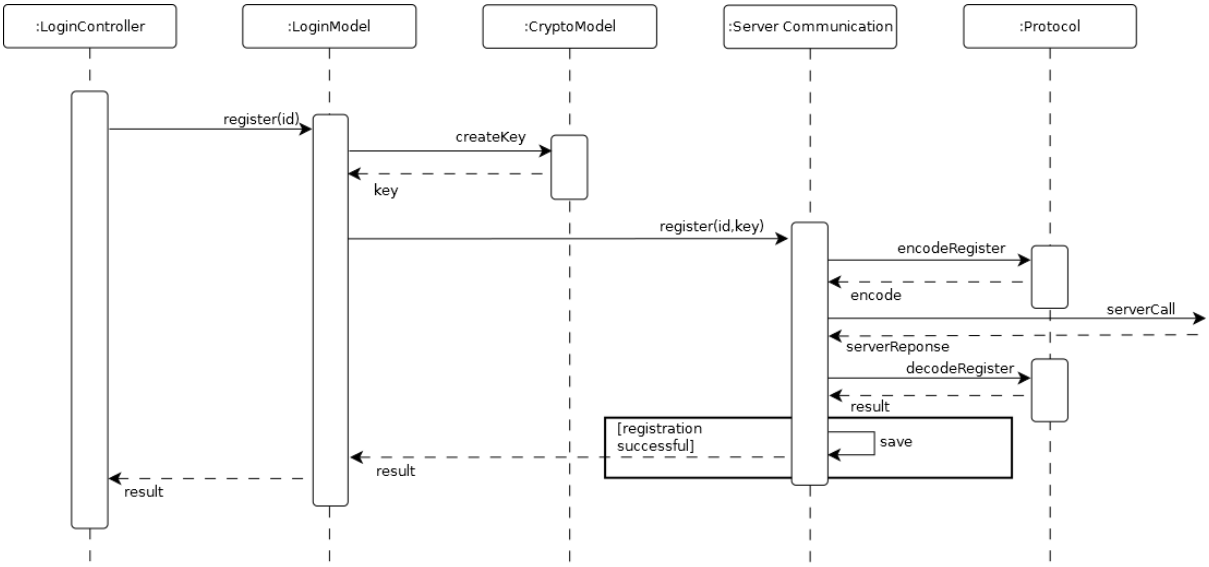


Aus Server-Sicht:

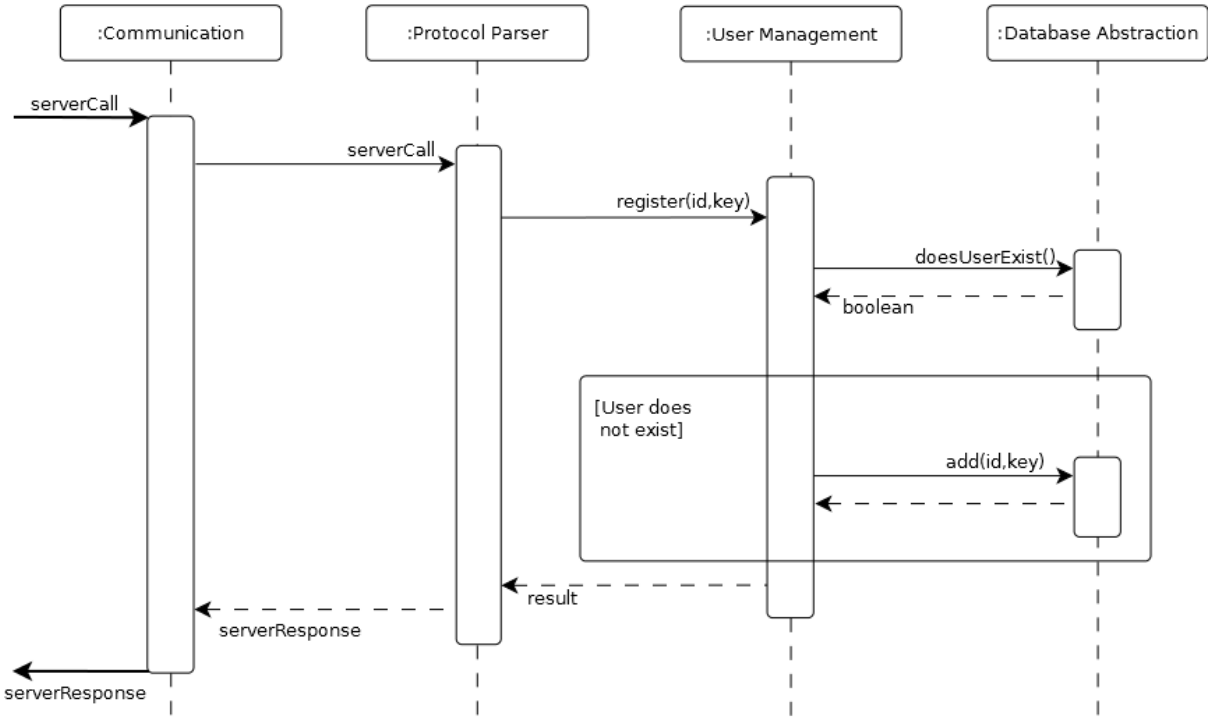


Registrieren eines neuen Nutzers

Aus Client-Sicht:

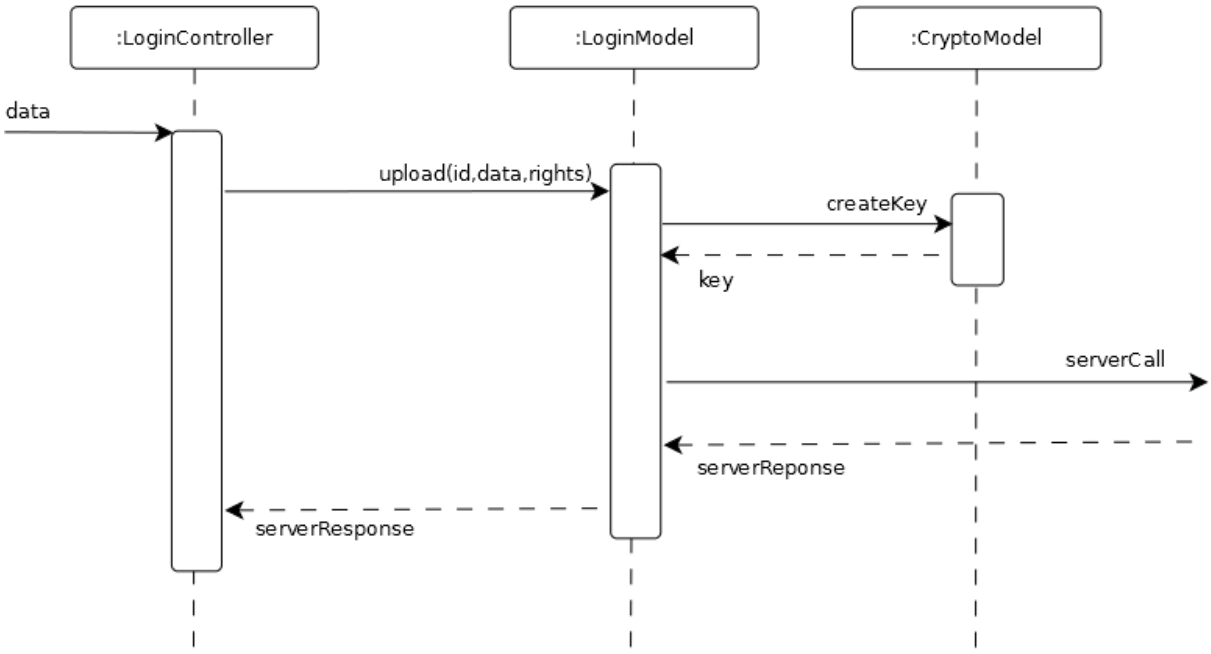


Aus Server-Sicht:

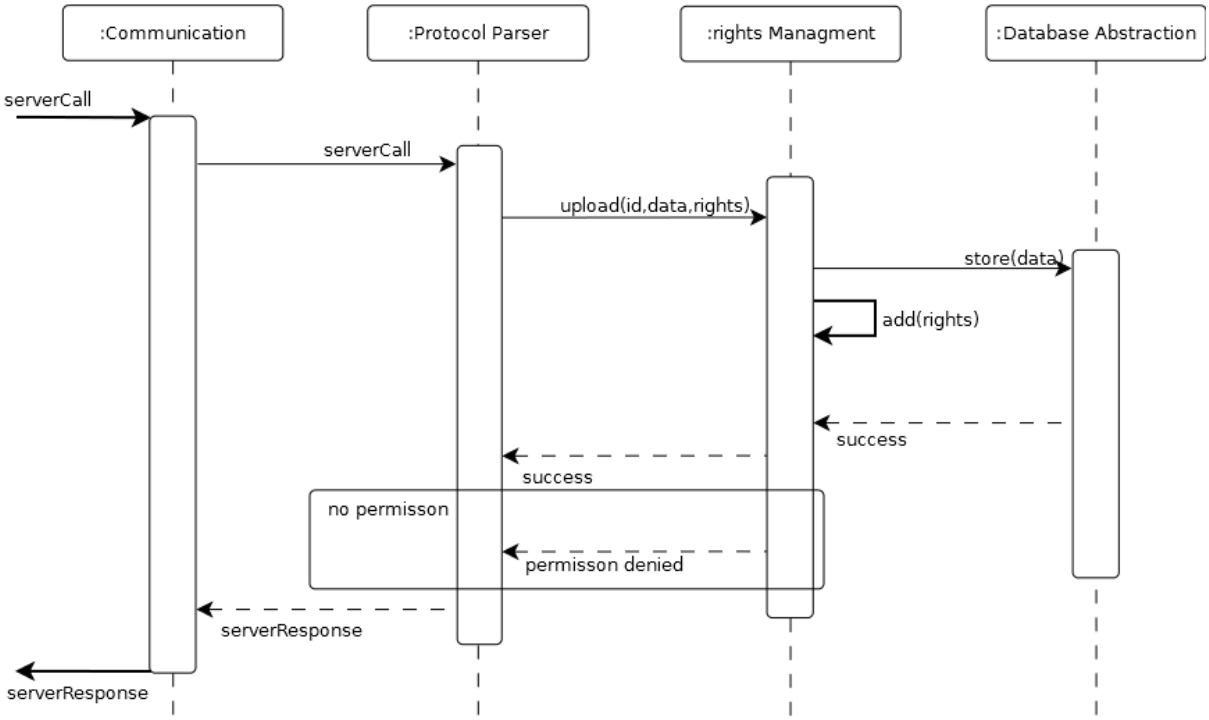


Hochladen eines neuen Datensatzes

Aus Client-Sicht:



Aus Server-Sicht:



Nutzung der Pattern

Bei der Erstellung dieses Dokuments wurden einige von Sommerville[2] und Buschmann[3] beschriebenen Pattern ausgelassen, da sich diese unserer Meinung nach nicht für „AtmoCalc“ anbieten.
Das Repository-Pattern wurde nicht genutzt, da „AtmoCalc“ nur über eine Web-Applikation nutzbar sein soll, es sollen also keine mobilen Apps oder Desktop-Anwendungen existieren. Der Server kann also auf die Web-Applikation angepasst sein und muss nicht in Bezug auf verschiedene Systeme generalisiert werden.
Ebenfalls nicht genutzt wurde das Microkernel-Pattern. Dies war nach unserer Auffassung für die mögliche Nutzung zu komplex und hätte die Komponenten, bei denen es eingesetzt worden wäre, unnötig verkompliziert. Auch muss das System nicht auf verändernde System-Requirements reagieren.
Weiterhin wurde das Broker-Pattern nicht genutzt, da das System laut den Requirements nicht auf mehreren Servern laufen soll.
Alle anderen, hier nicht aufgeführten Pattern, wurden aus ähnlichen Gründen vernachlässigt. Das Ziel dieses Dokumentes ist es, die von den Requirements vorgegebenen Funktionen des Systems in möglichst einfacher und logischer Form zu vermitteln; sonstige Pattern erschienen uns dafür nicht passend.

[2] Ian Sommerville. *Software Engineering*. Pearson Studium, 2012.

[3] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern Oriented Software Architecture: A System of Patterns*. Wiley, 1996.