POC

Proof of Concept

Semantische Repräsentation von Dokumenten durch den Einsatz der Werkzeuge: Protégé, Java, Jena, Jersey, Fuseki, Google Apps Script, App Engine und der Google Compute Engine

Erstellt von:
Kristjan Alliaj, Mateos Alliaj, Björn Zimmermann, Ivan Kurtovic

Hochschule der Medien Stuttgart
Master Wirtschaftsinformatik (WI3)
Modul: Technische Grundlagen Cloudbasierter Internet-Anwendungen
Wintersemester 2016/2017

Modulverantwortlicher:
Prof. Dr. Christian Rathke, Hochschule der Medien Stuttgart

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In der „Challenge 2 - Projekt Semantic Representation“ des Moduls “Technische Grundlagen Cloudbasierter Internetanwendungen“ ist die Herausforderung, eine Unterhaltung zwischen zwei Mitarbeitern so zu formalisieren, dass es einem Computer möglich wird aus diesem Gespräch selbstständige Handlungen abzuleiten und diese „zeitnah“ auszuführen. Konkreter soll es dem Computer möglich sein, den Gesprächspartnern Dokumente zur Auswahl zu stellen über die sie gerade gesprochen haben.

Diese Projektdokumentation wurde im Rahmen des Moduls Technische Grundlagen Cloud Computing angefertigt. Die Leistung dieses Moduls erfolgt durch ein Software-Projekt, indem zunächst ein Technologie-Scouting betrieben und anschließend ein Prototyp implementiert werden müssen. In dieser Dokumentation werden der Projektablauf sowie die Umsetzung des Prototyps festgehalten.

Die Dokumentation besteht aus 5 Kapiteln. Nach der Einleitung folgt in **Kapitel 2** eine Projektbeschreibung, in der die Ausgangssituation des Projektes erläutert, das Projektteam und die Anforderungen an das Projekt vorgestellt werden. Als nächstes werden in **Kapitel 3** die technischen Grundlagen, wie etwa die Architektur des Gesamtsystems beschrieben. In **Kapitel 4** wird beschrieben, wie der Prototyp umgesetzt wurde. Dazu werden die Auswahl der Technologien, der Zweck und die Architektur des Prototyps vorgestellt. Die Dokumentation wird in **Kapitel 5** mit einem Fazit und Ausblick abgeschlossen.

2. Projektbeschreibung

In diesem Kapitel wird zunächst die Ausgangsituation des Projektes beschrieben sowie das Projektteam vorgestellt. Anschließend wird dargestellt in welchem Teil der Architektur unsere Challenge ansetzt. Als nächstes werden die technischen Rahmenbedingungen erläutert. Dieses Kapitel dient dazu, die Ziele und Bedingungen des Projektes zu veranschaulichen.

2.1 Ausgangssituation

Das Projekt ist Gegenstand des Moduls: „Technische Grundlagen Cloud Computing“ in dem Masterstudiengang Wirtschaftsinformatik an der Hochschule der Medien im Wintersemester 2016/2017. Ziel des Projektes ist es, im Rahmen drei verknüpfter Module (1. Technische Grundlagen Cloudbasierter Anwendungen, 2. Sciene Lab, 3.Development Lab) eine Cloud-Applikation durch die Zusammenarbeit mehreren Teams zu entwickeln. Die Projektzeit beträgt 2 Semester. Das Projekt hatte seinen Kick-Off im Modul Technische Grundlagen von Cloud Applikationen (WS 16/16).

Der Auftraggeber ist der Chef eines fiktiven global agierenden Unternehmens, das eine neue Applikation auf Google-Hangout-Basis für die Verbesserung Projektkommunikation entwickeln lassen möchte. Das Zielsystem soll Zeit sparen, indem es spezifische Dokumente mit verschiedenen Datentypen während eines Video-Gespräches zum Öffnen vorschlägt. Die Dokumente liegen in einem firmeneigenen Dokumentenspeicher (GoogleDrive) und werden dem Nutzer über die Google-Hangout-Oberfläche vorgeschlagen. Der Nutzer kann die Dokumente dann öffnen und mit anderen Nutzern teilen.

Zur Umsetzung dieses Systems ist eine Machtbarkeitsstudie notwendig gewesen. Dazu wurden fünf Teilaufgaben (Challange’s) definiert, die jeweils von vier- bis fünfköpfigen Teams übernommen wurden. In jeder dieser Teilaufgaben sollten jeweils das Thema beschrieben, ein Technologie-Scouting durchgeführt sowie ein Prototyp erstellt werden. Anhand des Prototyps soll die Machbarkeit der Teilprobleme nachgewiesen werden. Die Implementierung des Systems folgt im 2. Semester im

Rahmen des Modules Development Lab. Im Folgenden werden die 5 Teilaufgaben bzw. Challenge's vorgestellt.

- **Challenge 1: Speech Tokenizer**
  Aus der Unterhaltung müssen z.B. auf Basis der Google Speech API Tokens abgeleitet werden, die eine Identifikation von Gesprächsthemen möglich machen.

- **Challenge 2: Repräsentation der Projekt- und Dokumentstrukturen sowie deren Beziehungen zu den Nutzern**
  Die in den Projektdokumenten dargestellte Information muss hinsichtlich ihrer Bedeutung für den Projektkontext aufbereitet werden. Die Frage, welche inhaltliche Funktion ein Dokument bezüglich eines Projekts besitzt, muss geeignet repräsentiert werden, um mittelbar die thematische Zuordnung des jeweiligen Dokuments zu einem Gesprächskontext zu ermöglichen.
  Technisch viele Möglichkeiten, z.B. in Google Cloud Storage; für die Inferenz könnte ggf. Drools verwendet werden.

- **Challenge 3: Event-basierte Integration**
  Die Verbindung zwischen Spracherkennung, Dokumentennutzung usw. erfordert die Repräsentation von nennenswerten Änderungen des Zustands des Gesamtsystems durch Events. Diese Events müssen interpretiert werden, und es sind auf Basis erkannter Ereignismuster neue Ereignisse (sog. Complex Events) abzuleiten, die schließlich in Aktivitäten resultieren, wie z.B. der Auswahl eines anzuzeigenden, relevanten Dokuments.

- **Challenge 4: Lernendes System**
  Mit der Zeit sollte sich das System bei der Präsentation relevanter Dokumente verbessern. Vom Nutzer verwendete, d.h. ausgewählte und/oder geöffnete Dokumente korrespondieren mit den sprachlichen Äußerungen der Gesprächspartner. Diese Zusammenhänge können mit der Zeit gelernt und für die Auswahl zu präsentierender Dokumente genutzt werden.

- **Challenger 5: User Experience**

  Das Gesamtsystem benötigt eine Komponente, die eine adäquate Visualisierung von und die Interaktion mit Recommendations ermöglicht.

2.2 Projektteam

Unserem Team wurde die Challenge 2 zugewiesen. Das Modul Technische Grundlagen Cloud Computing besteht aus 5 Meilensteinen. Die konkreten Anforderungen der **Module 2 und 3(SS 2017)** sind zum jetzigen Zeitpunkt noch unbekannt und werden erst im Sommersemester 2017 in den zwei weiteren Modulen: „Development-“ und „Scienec-“ Lab, gestellt.

Unser Team setzt sich aus den folgenden vier Mitgliedern zusammen: Kristjan Alliaj, Mateos Alliaj, Björn Zimmermann und Ivan Kurtovic. Die Verantwortlichkeiten haben wir nach den individuellen Kompetenzen des Teams aufgeteilt (siehe Tabelle 1):

Komponente	Rolle	Verantwortliche
Alle	Projektmanagement	Mateos Alliaj
Inference Engine	Entwickler	Mateos Alliaj, Kristjan Alliaj
KnowledgeBase- + Indexer-Komponente	Knowledge Engineer	Björn Zimmermann, Ivan Kurtovic

Tabelle 1: Verantwortlichkeiten

2.3 Anforderungen

Mit Use Case Diagrammen werden die Fragen beantwortet was genau das System im Endeffekt leisten sollte. Die Use Cases (siehe Abbildung 1 und Abbildung 2) wurden nach den Systemanforderungen angepasst, um diese übersichtlich für uns als Entwickler darzustellen. Aber auch für die Stakeholder sind die Use Cases wichtig, um die Anforderung nachvollziehen zu können. Eine Beschreibung des ersten Use Case ist in Tabelle 2 und die Beschreibung des zweiten Use Case ist in Tabelle 3 vorzufinden.

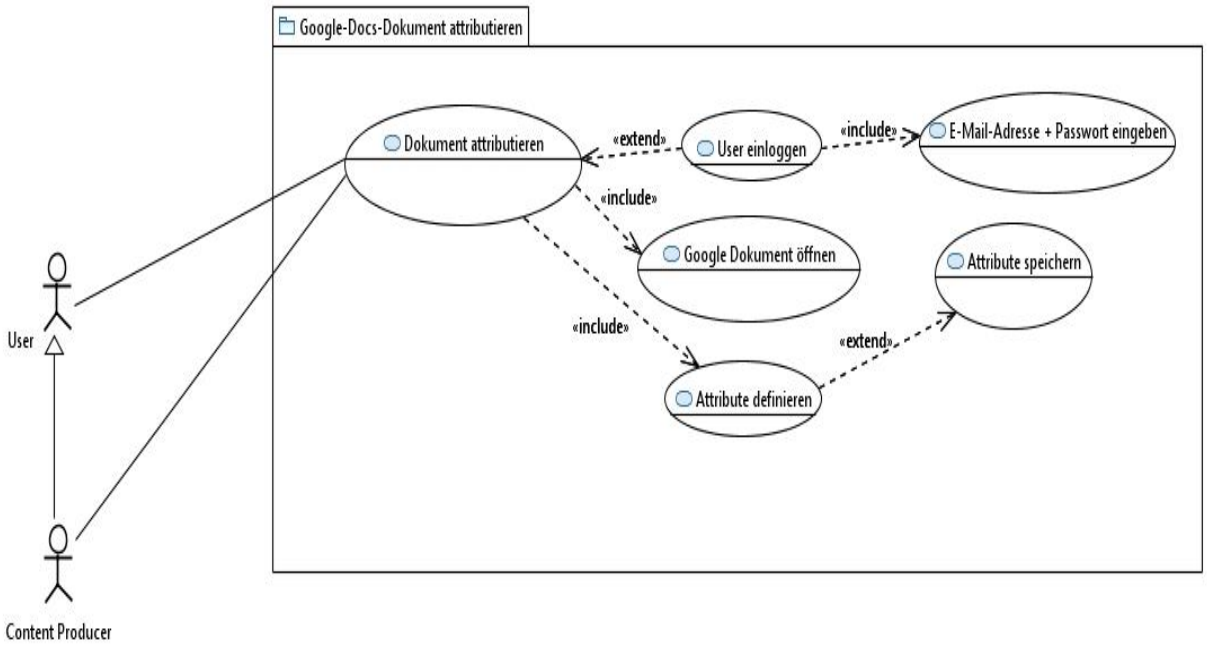


Abbildung 1: Use Case 1

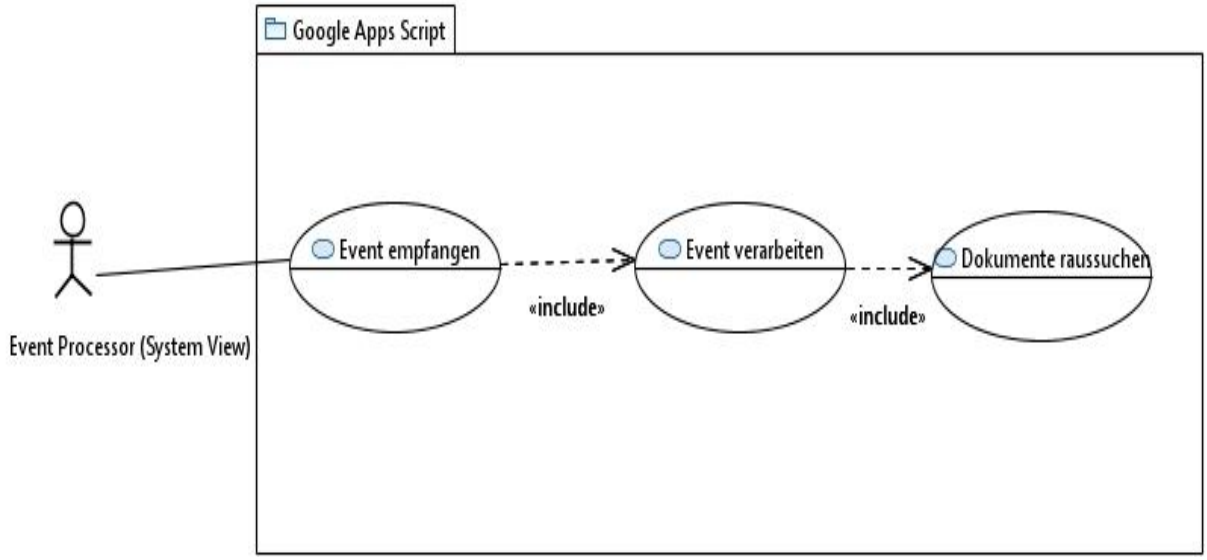


Abbildung 2: Use Case 2

Name
Google-Docs-Dokument attribuieren
Ziel
Ein User möchte ein Google-Docs-Dokument attribuieren
Akteure
User
Auslöser

User
Ergebnis
Das Google-Docs-Dokument wurde attributiert und gespeichert.
Eingehende Daten
Metadaten, Strings, Keywords
Normalablauf
1. Der User loggt sich ein. 2. Der User navigiert zum Bereich Google-Docs 3. Der User gelangt mit Hilfe eines Buttons zur Funktion „Google-Docs-Dokument öffnen“. 4. Der User gelangt mit Hilfe eines Buttons zur Funktion „Google-Docs-Dokument attribuieren“. 5. Der User attribuiert das Google-Docs-Dokument mithilfe der Docs-Erweiterung. 6. Der User speichert das Google-Docs-Dokument unter einem neuen Namen ab. 7. Das Google-Docs-Dokument wurde attributiert und gespeichert.
Vorbedingungen
I. Der User muss eingeloggt sein. II. Der User navigiert zum Bereich Google-Docs III. Der User gelangt mit Hilfe eines Buttons zur Funktion „Google-Docs-Dokument öffnen“. IV. Der User gelangt mit Hilfe eines Buttons zur Funktion „Google-Docs-Dokument attribuieren“.
Nachname
I. Die Modifikationen (Attributierung) des Google-Docs-Dokuments wurde gespeichert. II. Dem Google-Docs-Dokument wurde automatisch der Autor der letzten Änderung hinzugefügt. III. Dem Google-Docs-Dokument wurde ein automatisches Änderungsdatum hinzugefügt. IV. Der Knowledge Base (Ontology) wurden neue Attribute hinzugefügt
Nachbedingungen im Sonderfall Sollte das zu erstellende Google-Docs-Dokument bereits existieren, wird dieses nicht angelegt und der User erhält einen Warnhinweis.

Tabelle 2: Use Case 1

Name
Event empfangen
Ziel
Eingehender Event des Event-Processor-Systemteils soll vom Google Apps Script-Systemteil empfangen, verarbeitet und rausgesucht werden
Akteure
Event Processor, Google Apps Script
Auslöser
Event Processor
Ergebnis
Eingehender Event des Event-Processor-Systemteils wurde vom Google Apps Script-Systemteil empfangen, verarbeitet und rausgesucht
Eingehende Daten
Tokens, Strings, Keywords
Normalablauf
1. Event (Dokumenten-Anfrage) geht bei Google Apps Script-Systemteil ein 2. Event (Dokumenten-Anfrage) wird von dem Google Apps Script-Systemteil verarbeitet 3. Abhängig von dem eingehenden Event werden von dem Google Apps Script-Systemteil Dokumente rausgesucht.
Vorbedingungen
I. Der User muss eingeloggt sein. II. Das eingehende Event enthält Token-Keywords, welche verarbeitet werden können
Nachbedingungen I. Abhängig von dem eingehenden Event werden von dem Google Apps Script-Systemteil Dokumente rausgesucht
Nachbedingungen im Sonderfall Sollte das Event keine passenden Tokens enthalten, dann werden keine Dokument rausgesucht

Tabelle 3: Use Case 2

3. Technische Grundlagen

In diesem Kapitel werden die technischen Hintergründe des Projektes beschrieben. Es beginnt mit einer Beschreibung der Gesamtarchitektur. In dieser Gesamtarchitektur wird dargestellt, an welcher Stelle unser Projekt ansetzt, d.h. welche Komponenten der Gesamtarchitektur für die Semantische Repräsentation relevant sind.

3.1 Architektur des Gesamtsystems

Es sind für die erfolgreiche Umsetzung des Zielmoduls vier Einzelkomponenten zu realisieren, und zwar die Inference Engine, der Indexer, die Knowledge Base und die Document Base (siehe Abbildung 3). Im Rahmen des Moduls Technische Grundlagen Cloud Computing wird das Funktionsschema dies mit dieser Fallstudie und durch einen auf Github abgelegten Prototyp beschrieben. Für die weiteren Komponenten der Systemarchitektur sind die restlichen vier Teams zuständig. Das Ziel des Moduls Development Lab im zweiten Semester (SS 2016) wird es sein, die Teillösungen sämtlicher Themen zu einer Gesamtlösung zusammenzuführen.

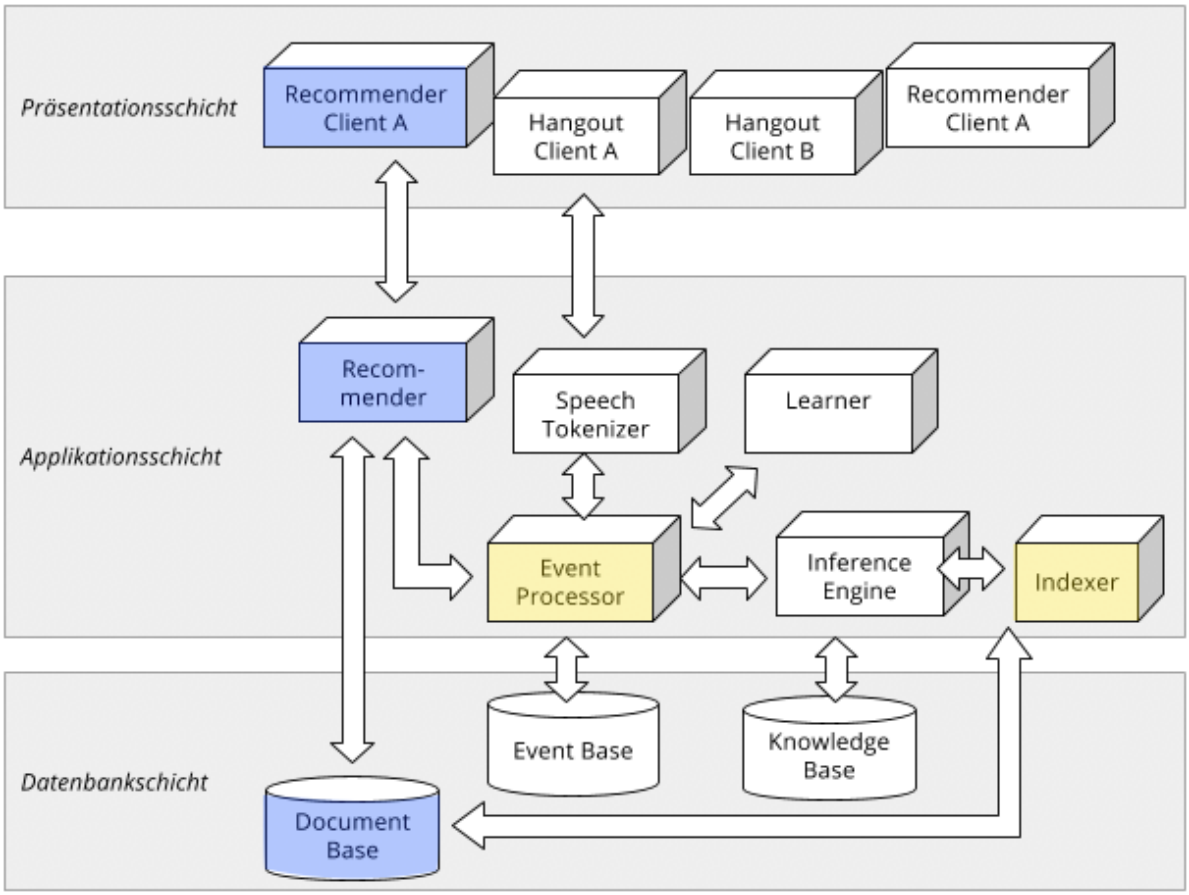


Abbildung 3: Systemarchitektur

3.2 Technische Rahmenbedingungen

TR1: Einzusetzende Systeme bzw. Dienste sind Google Hangout, Google Drive, Google Cloud Storage, Google Compute Engine, Google Container Engine, Google App Engine, Google Web Toolkit.

TR2: Für die Integration (Nutzung und Bereitstellung) von Diensten sind Web Services nach dem REST-Ansatz zu nutzen. Verfügbare Frameworks, die dies im Java-Umfeld erlauben, sind Jersey, RestEasy, Restlet und Google Cloud Endpoints.

TR3: Für die Ereignisverarbeitung sowie die Inferenz im Allgemeinen ist vorzugsweise Drools zu verwenden. Alternativ kann auch Esper genutzt werden.
Für die Persistenz von Daten kann wahlweise mit Google Cloud SQL eine relationale Datenbank oder eine der NoSQL-Alternativen wie Cloud Storage oder Appengine Datastore von Google verwendet werden. Dies ist ggf. abhängig von Anforderungen der anderen integrierten Systeme (siehe etwa Drools / Drools Fusion).
TR4: Die Anbindung von Datenbanken kann wahlweise durch Persistenzframeworks wie etwa Hibernate, JDO, anbieterspezifischen Frameworks oder eigenentwickelten Datenbank-Mappern unter Einsatz von Java Database Connectivity (JDBC) erfolgen.

TR5: Für die Dokumentation von Software-Strukturen ist die Unified Modeling Language (UML) zu verwenden.

TR6: Für die Nutzung von Cloud-basierten Datenbanken müssen die Gruppen ggf. pro Gruppe mindestens ein Benutzerkonto bei diesen Diensten anlegen. Dies ist etwa bei Google Cloud SQL und Google Cloud Storage der Fall.

TR7: Verwendete Programmiersprachen: Sofern eine Java API der verwendeten Dienste und Komponenten verfügbar ist, ist diese zu nutzen. Ergänzend kann auch auf Javascript zurückgegriffen werden.

3.3 Ontologie-Mindmap

Wir haben zunächst in einer Mindmap (siehe Abbildung 4) die benötigte Ontologie entworfen.

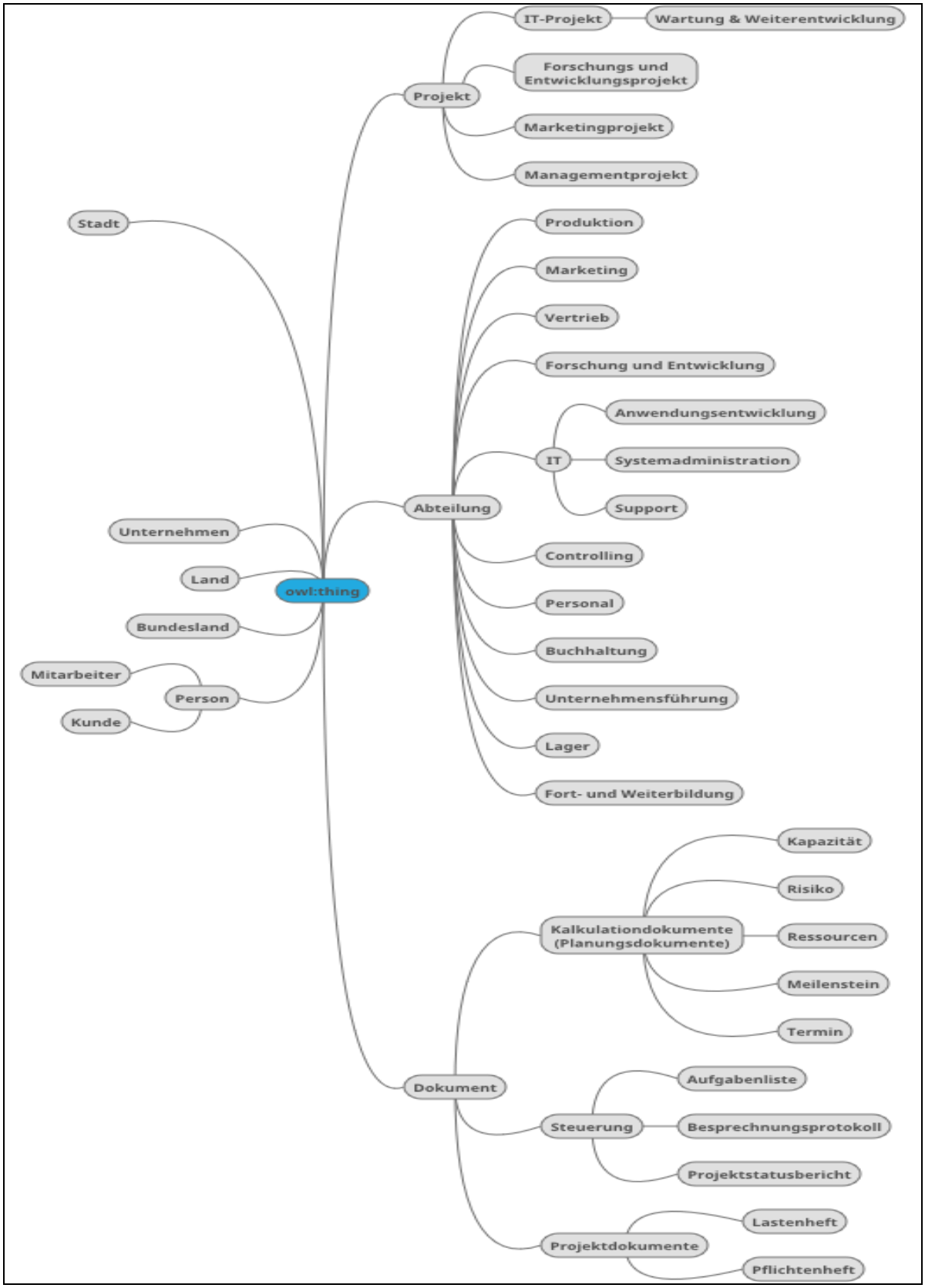


Abbildung 4: Ontologie-Mindmap

4. Umsetzung des Prototyps

In diesem Kapitel wird beschrieben, wie der Prototyp zum Machbarkeitsnachweis umgesetzt wurde. Dabei wird zunächst die Auswahl der Werkzeuge vorgestellt und begründet. Als nächstes wird auf die Implementierung des Prototyps eingegangen. Dabei wird anhand verschiedener Szenarien gezeigt, dass der Prototyp die an das Projekt gestellten Anforderungen erfüllt. Außerdem wird die Architektur des Prototyps dargestellt, um den Systemablauf dieses nachvollziehen zu können. Des Weiteren wurde versucht, weitere Features einzubauen, bei denen es jedoch einige Probleme in der Implementierung gab. Diese Probleme werden zum Ende dieses Kapitels beschrieben.

4.1 Technologieauswahl

In diesem Kapitel werden die ausgewählten Technologien und Werkzeuge vorgestellt, die wir zur Umsetzung des Projekts verwendet haben. Außerdem wird die Auswahl der Technologien und Werkezuge begründet. Dazu mussten für die Definition der Ontologie diverse Beschreibungssprachen miteinander verglichen werden. Als nächstes wurden verschiedene Editoren evaluiert, welche die Modellierung von Ontologie anbieten. Anschließend wurden einige Frameworks evaluiert, mit denen es möglich ist, Ontologie zu manipulieren. Die Vergleiche sämtlicher Technologien, Werkzeuge und Frameworks wurden anhand verschiedener Kriterien evaluiert. Die Auswahl der Kriterien wurde nach einer umfangreichen Recherche relevanter Kriterien getroffen.

4.1.1 Allgemein

Um eine Best-Practice zu ermitteln, wie und vor allem mit welchen Tools, wir die semantische Dokumenten Repräsentationen, realisiert werden kann. Betrachten wie drei unterschiedliche Ebenen: Die Beschreibungssprachen (XML, RDF, OWL), Ontologie-Editoren (Protege, etc.) und semantische Frameworks (Jena, etc.). Jeder dieser Betrachtungsweisen wiesen wir Kriterien in Ergonomischer, Ökonomischer und Technischer-Art zu. Die Evaluierung der Werkzeuge wurde in separaten Excel-Tabellen durchgeführt. Diese sind in dieser Dokumentation nicht enthalten, sondern lediglich eine Zusammenfassung der Ergebnisse der jeweiligen Evaluationen.

Werkzeug	Kriterien
XML, RDF, OWL	• Angabe von Relationen • Serialisierung • Semantik (Bedeutung) von Dokumenten • Schema • Beschreibungsebenen (anreichern mit Meta-Daten) • Vokabular • HTML Einbettung • Klassenbildung • Math. Operatoren (Mengenlehre)
Ontologie-Editoren	• Allgemein • Kompatibilität • Wissensrepräsentation und methodologische Unterstützung • Inference Services • Wirtschaftlichkeit • Entwicklung
Semantische Frameworks	• Allgemein • Inference Engine • Wirtschaftlichkeit

Tabelle 4: Kriterien: Technologie-Scouting

4.1.2 Evaluierung der Beschreibungssprachen

Trotz der hohen formalen Ausdrucksfähigkeit von RDF und XML kommt für uns nur OWL, konkret OWL-Full, als Beschreibungssprache für Ontologie in Frage, da andere Ausprägungen, OWL-Lite und OWL-DL kompatibel mit dem RDF Schema sind. Was für OWL spricht ist, dass das Vokabular von RDF maßgeblich erweitert wird. Zum Beispiel was die Beziehungen von Klassen zueinander sind oder was die Gleichheit von Objekten angeht. Des Weiteren kann man in OWL definieren, wie es geht und wie es nicht geht. Konkret bedeutet das, dass mittels OWL symmetrische, transitive sowie inverse Beziehungen abgebildet werden können. Auch die Deklaration von Domänen ist gegeben, in der wir in unserem Projekt zum Beispiel eine Abteilung des Auftraggeber-Unternehmens zuweisen können. Ein weiterer

Punkt ist, dass man mittels OWL durch Klassen Schnittmengen, Vereinigungen und Komplemente abbilden kann.

4.1.3 Evaluierung der OWL-Editoren

Zusammenfassend lässt sich sagen, Protégé und Swoop im Gegensatz zu Ontostudio open-source sind und somit keine Nutzungs- oder Lizenzgebühr anfallen. Zudem sind alle betrachteten Werkzeuge plattformunabhängig. Protégé und Ontostudio verwenden für die Speicherung von Ontologien Datenbanken, SWOOP nicht. Protege bringt eine Hauseigene Inference Engine mit, bei SWOOP und Ontostudio können externe angebunden werden. Die Oberfläche von Protégé wirkt intuitiv und benutzerfreundlich und ist somit mehr ein grafisches Ontologie-Werkzeug. Für Protégé spricht ebenfalls eine umfassende Dokumentation, die den Einstieg in die Arbeit mit dem Werkzeug beschreibt, sowie Szenarien und Beispiel-Topologien. Wir haben uns für Protege nicht nur aus den oben genannten Gründen entschieden, untermauert wird die Entscheidung, dass das Entwicklernetzwerk, dass Protege entwickelt hat, von der Universität in Stanford betreut wird und enorme Nutzerzahlen aufweist. Trotz den verschieden Alternativen passt Protege als Ontologie-Editor am besten zu den Projektanforderungen.

4.1.4 Evaluierung der semantischen Frameworks

Zusammenfassend ist zu sagen, dass Apache Jena ein umfangreiches semantisches Framework darstellt, die OWL-API sich auf die Erstellung und Manipulation von Ontologie beschränkt. Jena bietet eine hauseigene Inference Engine API und kann zusätzlich durch weitere Reasoners, wie zum Beispiel Pallet erweitert werden. Die OWL-API bietet ein Interface an, um bekannte Reasoner, wie beispielsweise Fact++ einzubinden. Für Jena spricht eine eigene Regel-Engine, die es ermöglicht spezifische Ableitungsregeln zu definieren. Jena ist ein sehr umfangreiches Framework im Gegensatz zur OWL-API. Wir werden mit Jena arbeiten, da Jena den kompletten “Layer-Cake” abdeckt und damit sehr geeignet ist, um Dokumente semantisch zu repräsentieren. Die OWL-API werden wir uns dennoch näher anschauen, vielleicht gerade um die Machbarkeit im Rahmen eines Prototyps zu demonstrieren.

4.1.5 Zusammenwirkung der Technologien

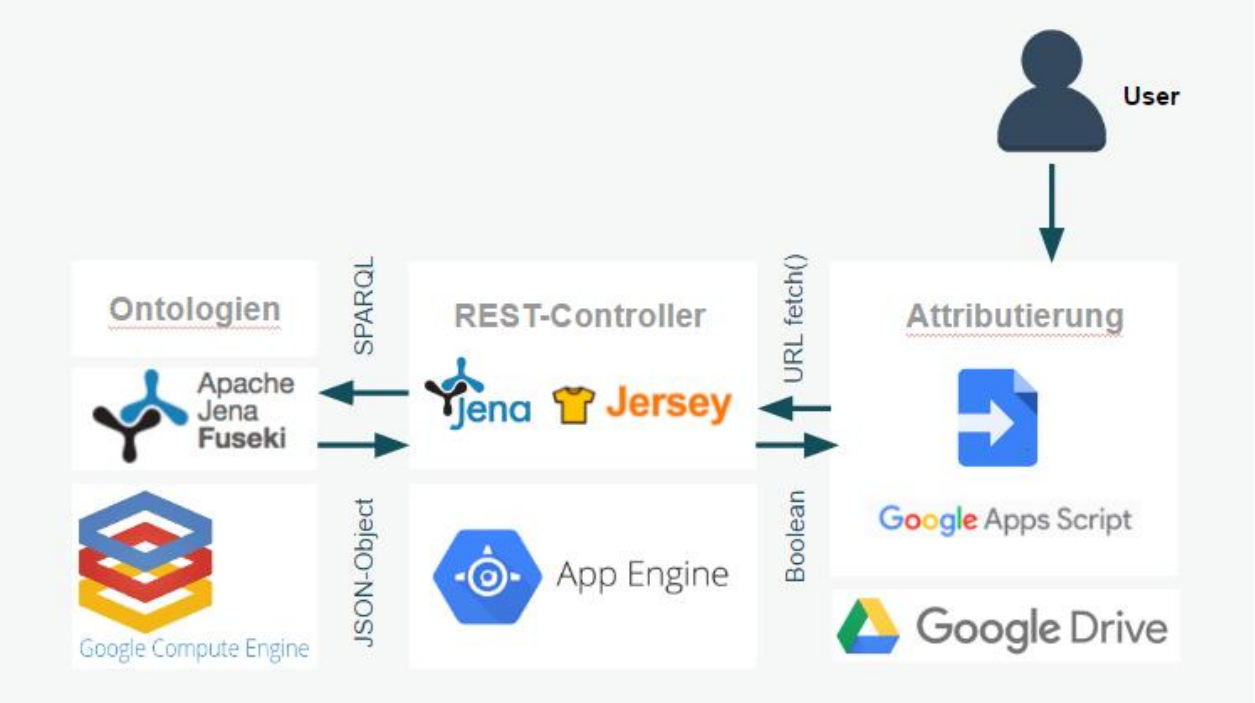


Abbildung 5: Einsatz der Technologien - Anwendersicht

Abbildung 5 zeigt den Endpoint für das Google Drive Addon an. In diesem kann der Nutzer eigene Metadaten zu dem geöffneten Dokument hinterlegen, diese. bearbeiten und speichern. Beim Speichern wird die eine “URL-Fetch”-Methode gestartet, diese transportiert die eingegeben Formulardaten des Nutzers über das Nachrichtenformat JSON und übergibt die Daten via eines HTTP Request an den Jersey Restservice, der in unserem Modell auf der Google App Engine ausgeführt wird.

Durch das semantische Framework Apache Jena verarbeitet wir die erhaltenen Daten weiter und fügen diese in dynamischen SPARQL. Im Anschluss feuert wir die Query. Über einen REST-Service, also über einen HTTP-Request nimmt der Apache Fuseki Server diese entgegen.

Der Apache Fuseki Server ist unser Datenbankserver, in dem die Knowledgebase (A-Box und T-Box) hinterlegt sind. Nach erhalt des Request wird auf dem Fuseki Server die SPARQL ausgeführt, die die dann wiederum je nach Parameterübergabe die Knowledgebase manipuliert. Ausgeführt wird der Jena Fuseki Server auf der

Google Compute Engine. Auf dieser Läuft eine virtuelle Serverinstance mit dem Betriebssystem Ubuntu.

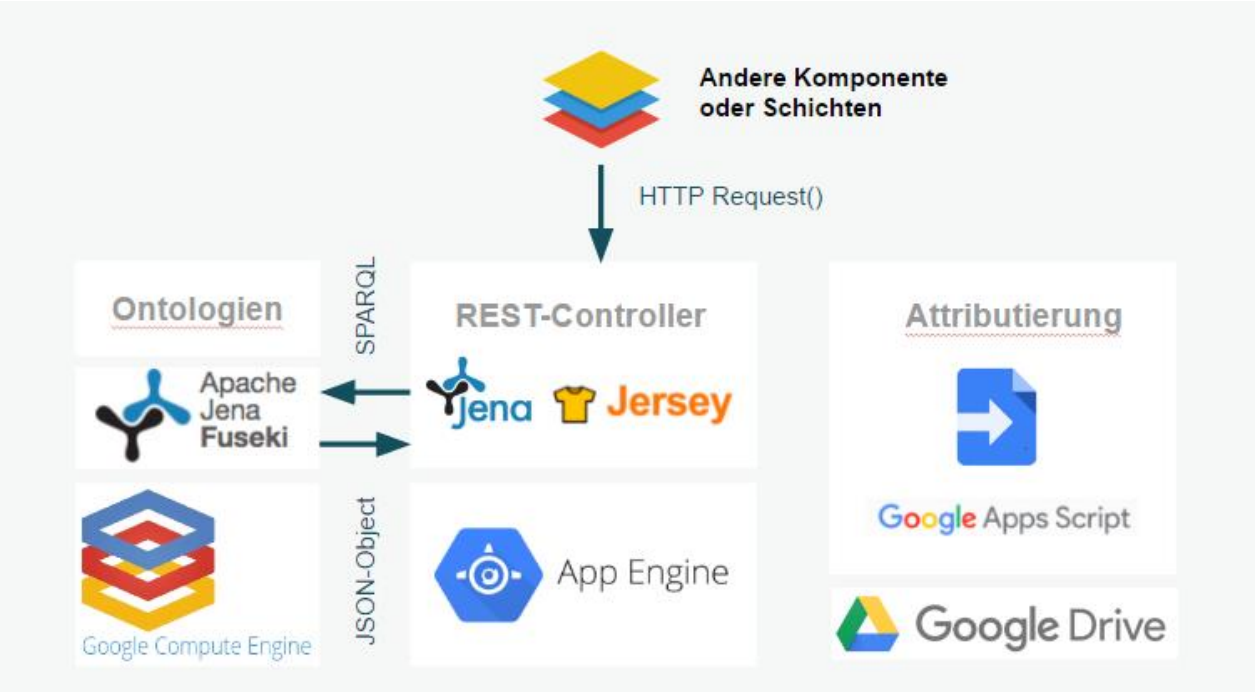


Abbildung 6: Einsatz der Technologien - Schichten

Als Datengrundlage für die semantische Repräsentation von Dokumenten musste eine Ontologie erstellt werden. Diese wurde im OWL-Format über den OWL-Editor Protégé erzeugt, um mehr Details und Einschränkungen definieren zu können. Die Ontologie wurde anschließend in Java mit entsprechenden Apache Jena API´s verarbeitet. Hierdurch konnten Anfragen über SPARQL übermittelt werden, um Informationen zu den in der Ontologie angelegten Dokument-Instanzen zu erhalten.

Über die lokale Verarbeitung in der Entwicklungsumgebung hinaus, kann die Ontologie auch auf einen lokalen Apache Jena Fuseki Server geladen werden. Anschließend kann der Fuseki-Server aus Java heraus angesprochen werden.

Das Google Apps Script dient der Attributierung von neuen Dokumenten, die in der Ontologie ergänzt werden sollen. Die dort stattfindende Attributierung wird über einen URL-Fetch übermittelt und kann durch REST-Schnittstellen in der Java-Anwendung getriggert werden.

Die REST-Schnittstellen wurden exemplarisch mit dem Jersey-Framework auf der Google App Engine getestet. Als Ausblick werden die REST-Schnittstellen erweitert, um die Erweiterung der Ontologie in der Java-Anwendung durch die Verknüpfung mit der Google Apps Script Attribuierung zu ermöglichen. Außerdem soll die Übermittlung von Daten auch an die anderen Schichten und Komponenten im gesamten semantischen Projekt über JSON-Objekte ermöglicht werden. Dies wird durch die offene REST-Methodologie ermöglicht.

4.2 Implementierung des Prototyps

Im Laufe des Wintersemesters 2016/2017 hat unser vierköpfiges Team für das Modul “Technische Grundlagen cloudbasierter Internet Anwendungen“ mehrere lauffähiges semantische Szenarien entwickelt und realisiert welche in den nachfolgenden Abschnitten beschrieben werden. Das Ziel der entwickelten Szenarien ist es die Machbarkeit zu beweisen sowie die Funktionsweise der Technologie zu veranschaulichen.

4.2.1 Szenario 1

Das erste Szenario veranschaulicht an einem fiktiven Beispiel eine Situation, bei der sich zwei Gesprächspartner über eine vor kurzem erstellte PowerPoint-Präsentation unterhalten und diese angezeigt wird. Im ersten Szenario wird die Knowledge Base (T-Box und A-Box), welche das ganze Domänenwissen abbildet mittels einer statischen SPARQL-Anfrage abgefragt. Konkret werden durch eine SPARQL-SELECT-Anweisung sämtliche Dokumente sowie deren Google Drive URLs mit dem Dateiformat PowerPoint-Präsentation (.pptx) vom November 2016 ausgegeben, welche in der Knowledge Base/Ontology zu finden sind. Hinter den Links der Google-Drive-URLs verbergen sich Projektdokumente, welche z.B. während eines Projektes erstellt wurden.

In Abbildung 7 ist die statische SELECT-Anweisung in SPARQL-Notation zu sehen, welche im ersten Szenario verwendet wurde. Die SPARQL-Anfrage enthält sowohl Angaben über die T-Box als auch über die A-Box, in der sich die Daten (Google-Drive-URLs etc.) befinden.

```
String sparql = "PREFIX foaf: <http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#>"
                + "     PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                + " PREFIX mebase: <http://www.semanticweb.org/bjorn/ontologies/2016/11/tb_cloud#>"
                + " SELECT ?Dokumentname ?URL "
                + " WHERE  { "
                + " ?x foaf:URL ?URL ."
                + " ?x foaf:Dokumentname ?Dokumentname ."
                + " ?x foaf:DokumentTyp 'PowerPoint-Präsentation (.pptx)' ."
                + " ?x foaf:ErstellugsDatum ?ErstellugsDatum ."
                + " Filter ( str(?ErstellugsDatum) > '2016-11-01T00:30:00' )"
                + "     }";
```

Abbildung 7: SPARQL SELECT-Anweisung des ersten Szenarios

Im Folgenden ist die A-Box der Knowledge Base/Ontology zu sehen, welche mittels SPARQL abgefragt wurde. In der A-Box befinden sich die konkreten Daten (Google-Drive-URLs etc.). In der T-Box sind abstrakten Regeln und Beziehungen der Domäne beschrieben.

```
<!-- http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument1 -->
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument1">
<DokumentTyp rdf:datatype="http://www.w3.org/2001/XMLSchema#string">PowerPoint-Präsentation (.pptx)</DokumentTyp>
<Dokument_hat_Autor rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Haruki_Murakami"/>
<Dokumentname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Testdokument1</Dokumentname>
<ErstellugsDatum rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2016-11-12T07:55:00</ErstellugsDatum>
<Status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Technische Realisierung</Status>
<URL
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">https://docs.google.com/document/d/1fOFkCM6wb_Ux4za45iQLlsqbVfTlceO
6eTMiZZwiy70</URL>
<Version rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</Version>
<gehoert_zu_Projekt rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#EcoLogics"/>
</owl:NamedIndividual>
```

In Abbildung 8 ist das Ergebnis der statischen SELECT-Anweisung in SPARQL-Notation zu sehen, die im ersten Szenario benutzt wurde. Die Ergebnisse sind zwei Google-Drive-URLs, hinter denen sich Projektdokumente befinden.

```
| Dokumentname       | URL                                                                                    |
| "Testdokument1"    | "https://docs.google.com/document/d/1fOFkCM6wb_Ux4za45iQLlsqbVfTlceO6eTMiZZwiy70" |
| "Marketingkonzept" | "https://drive.google.com/open?id=1tTbGP2YiQxUef__4StLVbtzr_5NpAIMwRhAYeEnWJw8"   |
```

Abbildung 8 : Ergebins der SELECT-Anweisung des ersten Szenarios

4.2.2 Szenario 2

Im zweiten Szenario wird die Knowledge Base (T-Box und A-Box), die das ganze Domänenwissen abbildet mittels einer statischen SPARQL-Anfrage abgefragt. Konkret werden durch eine SPARQL-SELECT-Anweisung sämtliche Google-Drive-Dokumente angezeigt, welche die Keywords ***Projektplan*** und ***Haruki Murakami*** enthalten. Es wird somit die Ontology nach **Dokumentenname** und **Autor** durchsucht.

In Abbildung 9 ist die statische SELECT-Anweisung in SPARQL-Notation zu sehen, welche im zweiten Szenario verwendet wurde. Die SPARQL-Anfrage enthält sowohl Angaben über die T-Box als auch über die A-Box, in der sich die Daten (Google-Drive-URLs etc.) befinden.

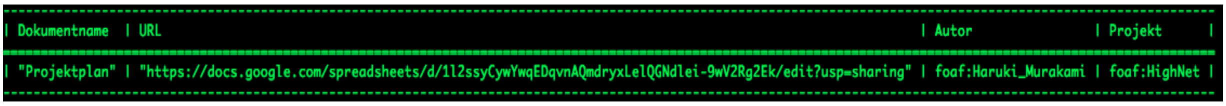
```
String sparql = "PREFIX foaf: <http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#>"
              + "     PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
              + " PREFIX mebase: <http://www.semanticweb.org/bjorn/ontologies/2016/11/tb_cloud#>"
              + " SELECT ?Dokumentname ?URL ?Autor ?Projekt "
              + " WHERE  { "
              + " ?Dokument foaf:URL ?URL ."
              + " ?Dokument foaf:Dokumentname ?Dokumentname ."
              + " ?Dokument foaf:Dokument_hat_Autor ?Autor ."
              + " ?Dokument foaf:gehoert_zu_Projekt ?Projekt."
              + " FILTER regex( str(?Autor), 'Haruki_Murakami' )"
              + " FILTER regex( str(?Dokumentname), 'Projektplan' )"
              + "}";
```

Abbildung 9: SPARQL SELECT-Anweisung des zweiten Szenarios

Im Folgenden ist die A-Box der Knowledge Base/Ontology zu sehen, welche mittels SPARQL abgefragt wurde. In der A-Box befinden sich die konkreten Daten (Google-Drive-URLs etc.). In der T-Box sind die abstrakten Regeln und Beziehungen der Domäne beschrieben.

```
<!-- http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Projektplan -->
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Projektplan">
<AenderungsDatum rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2016-01-15T15:08:00</AenderungsDatum>
<DokumentTyp rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Excel-Arbeitsmappe (.xlsx)</DokumentTyp>
<Dokument_hat_Autor rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Haruki_Murakami"/>
<Dokumentname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Projektplan</Dokumentname>
<ErstellugsDatum rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2016-01-12T10:38:00</ErstellugsDatum>
<URL
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">https://docs.google.com/spreadsheets/d/1l2ssyCywYwqEDqvnAQmdryxLel
QGNdlei-9wV2Rg2Ek/edit?usp=sharing</URL>
<Version rdf:datatype="http://www.w3.org/2001/XMLSchema#string">4</Version>
<gehoert_zu_Projekt rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#HighNet"/>
</owl:NamedIndividual>
```


In Abbildung 10 ist das Ergebnis der statischen SELECT-Anweisung in SPARQL-Notation zu sehen, welche im zweiten Szenario benutzt wurde. Das Ergebnis ist eine Google-Drive-URL, hinter der sich ein Projektdokument befindet.



Dokumentname	URL	Autor	Projekt
"Projektplan"	"https://docs.google.com/spreadsheets/d/1l2ssyCywYwqEDqvnAQmdryxLelQGNdlei-9wV2Rg2Ek/edit?usp=sharing"	foaf:Haruki_Murakami	foaf:HighNet

Abbildung 10: Ergebins der SELECT-Anweisung des zweiten Szenarios

4.2.3 Szenario 3

Im dritten Szenario wird veranschaulicht, wie eine manuelle Attributierung von Projektdokumenten funktionieren könnte. Jedes Google-Docs-Dokument wurde mit Hilfe eines Google-Apps-Scripts um ein Addon erweitert.

Das Addon erweitert Google-Docs-Dokumente um zusätzliche Auswahlfelder, die es ermöglichen Google-Docs-Dokumente manuell zu attribuieren. Die Attributierung ermöglicht es, Google-Docs-Dokumenten semantische Informationen zuzuweisen und somit diese inhaltlich für Computer/Maschinen verständlich zu machen. In Abbildung 11 ist das Google-Docs Addon zu sehen.

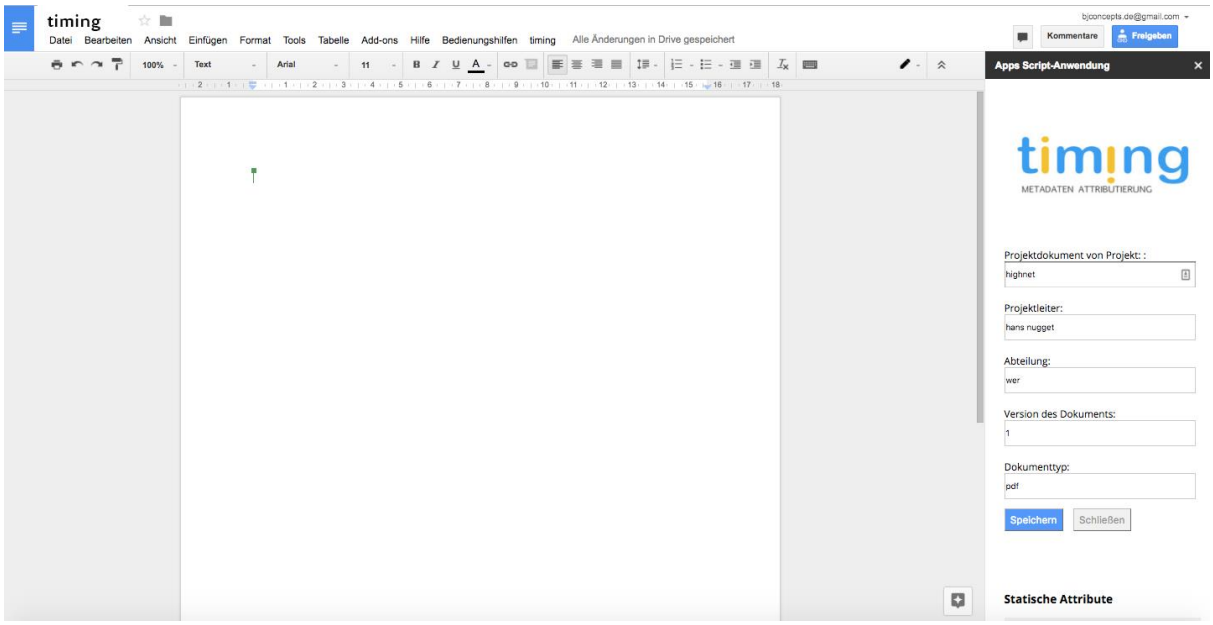


Abbildung 11: Google-Docs Addon

Folgende Attribute können durch das Google-Addon vergeben werden:

- **gehoert_zu_Projekt:** “HighNet”
- **Projektleiter:** “Hans Nugget”
- **Abteilung:** “Wer”
- **Version:** “1”
- **DokumentTyp:** “PDF”

4.2.4 Szenario 4

Im vierten Szenario wird eine doppelte SPARQL Anfrage durchgeführt. Zuerst wird die A-Box, welche das ganze Domänenwissen abbildet mittels einer statischen SPARQL-Anfrage um neue Dokumente erweitert. Konkret werden durch eine SPARQL-UPDATE-Anweisung neue Daten (Person, Dokumente, URLs etc.) in die Knowledge Base/Ontology hinzugefügt. Die Knowledge Base/Ontology enthält sozusagen “neues” Wissen und wächst in ihrem Umfang“. In Abbildung 12 ist die statische INSERT-Anweisung in SPARQL-Notation zu sehen, die im vierten Szenario verwendet wurde. Die SPARQL-Anfrage enthält sowohl Angaben über die T-Box als auch über die A-Box. Nach der Definition der Namensräume (PREFIX) wird über die SPARQL UPDATE- und WHERE-Befehle die Anfrage konkretisiert.

```
// Ergänzung der Ontologie um weitere Dokument-Instanzen
String sparql_insert = "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "
                + "PREFIX tb_cloud: <http://www.semanticweb.org/bjorn/ontologies/2016/11/tb_cloud#> "
                + "PREFIX ab_cloud: <http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#> "
                + "PREFIX foaf: <http://xmlns.com/foaf/0.1/> "
                + "PREFIX owl: <http://www.w3.org/2002/07/owl#> "
                + "INSERT DATA { "
                + "<http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#"
                + dokumentName
                + "> "
                + "a owl:NamedIndividual ; "
                + "ab_cloud:Dokumentname '"
                + dokumentName
                + "' ; "
                + "ab_cloud:DokumentTyp '"
                + dokumentTyp
                + "' ; "
                + "ab_cloud:Dokument_hat_Autor  <"
                + dokument_hat_Autor
                + "> ; "
                + "ab_cloud:ErstellugsDatum '"
                + erstellugsDatum
                + "^^xsd:dateTime' ; "
                + "ab_cloud:AenderungsDatum '"
                + aenderungsDatum
                + "^^xsd:dateTime' ; "
                + "ab_cloud:gehoert_zu_Projekt <"
                + gehoert_zu_Projekt
                + "> ; "
                + "ab_cloud:URL '"
                + dokumentURL
                + "' ; "
                + "ab_cloud:Version '" + dokumentVersion + "' ; " + " }";
```

Abbildung 12: SPARQL INSERT-Anweisung des vierten Szenarios

Im Folgenden ist die A-Box der Knowledge Base/Ontology zu sehen, die mittels SPARQL erweitert wurde.

ab_cloud:Aufgabenliste
ah_owl:NamedIndividual ;
ab_cloud:AenderungsDatum "2017-01-21T15:08:00^^xsd:dateTime"
ab_cloud:DokumentTyp "PowerPoint-Präsentation (.pptx)" ;
ab_cloud:Dokument_hat_Autor <file:///Users/mateos_alliaj/Documents/github/SemantischeRepresentation/Java/SemantischeRepresentation/Haruki_Murakami> ;
ab_cloud:Dokumentname "Aufgabenliste" ;
ab_cloud:ErstellugsDatum "2017-01-21T15:08:00^^xsd:dateTime" ;
ab_cloud:URL "https://drive.google.com/BeispielURL/4" ;
ab_cloud:Version "1" ;
ab_cloud:gehoert_zu_Projekt <file:///Users/mateos_alliaj/Documents/github/SemantischeRepresentation/Java/SemantischeRepresentation/HighNet> .

Nach der Erweiterung wird im selben Moment die Knowledge Base/Ontology durch eine SPARQL SELECT-Anfrage abgefragt (siehe Abbildung 13).

```
// Auswahl von Dokumentname und URL
String sparql_select = "PREFIX foaf: <http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#>"
                + "     PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                + " PREFIX mebase: <http://www.semanticweb.org/bjorn/ontologies/2016/11/tb_cloud#>"
                + " SELECT ?Dokumentname ?URL "
                + " WHERE  { "
                + " ?x foaf:Dokumentname ?Dokumentname ."
                + " ?x foaf:URL ?URL ." + "     }";
```

Abbildung 13: SPARQL SELECT-Anweisung des vierten Szenarios

In Abbildung 14 ist das Ergebnis der statischen SELECT-Anweisung in SPARQL-Notation zu sehen, die im vierten Szenario verwendet wurde. Das Ergebnis zeigt, dass der Ontology neue Instanzen hinzugefügt wurden.

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Dokumentname       | URL                                                                                                                |
===========================================================================================================================================
| "Marketingkonzept" | "https://drive.google.com/open?id=1tTbGP2YiQxUef__4StLVbtzr_5NpAIMwRhAYeEnWJw8"                                    |
| "Pflichtenheft"    | "https://docs.google.com/document/d/1lsy1JHKdjTsp0mGmnhN4LRTOBhDA3HiE47VaAE1dJ84/edit?usp=sharing"                 |
| "Projektplan"      | "https://docs.google.com/spreadsheets/d/1l2ssyCywYwqEDqvnAQmdryxLelQGNdlei-9wV2Rg2Ek/edit?usp=sharing"             |
| "Testdokument1"    | "https://docs.google.com/document/d/1fOFkCM6wb_Ux4za45iQLlsqbVfTlceO6eTMiZZwiy70"                                   |
| "timing"           | "https://drive.google.com/BeispielURL/4"                                                                           |
| "Aufgabenliste"    | "https://drive.google.com/BeispielURL/4"                                                                           |
-------------------------------------------------------------------------------------------------------------------------------------------
```

Abbildung 14: Ergebins der SELECT-Anweisung des vierten Szenarios

4.3 Architektur des Prototyps

In diesem Abschnitt wird dargestellt, wie der Prototyp funktioniert. Dazu wird zunächst die Ontologie beschrieben. Anschließend wird der Ablauf des Prototyps dargestellt. Dieser Abschnitt dient dazu, nachvollziehen zu können, wie der Prototyp technisch funktioniert. Eine detaillierte Beschreibung der Relationen zwischen den Klassen ist in aufgelistet.

4.3.1 Vorgehen bei der Ontologie

Wie in Abbildung 15 zu sehen ist, wird eine T-Box sowie eine A-Box benötigt, um unsere Knowledge-Base/Ontologie mit Hilfe des Ontologie-Editors Protege zu erstellen. Die T-Box beinhaltet allgemeine Aussagen und Axiome über Konzepte (Klassen) und Rollen. Die A-Box hingegen beinhaltet Aussagen über die Individuen von Klassen/Konzepten.

Damit die A-Box und T-Box in Protégé streng getrennt bleiben, dürfen die Instanzen auf der Instanzebene/Objektebene ausschließlich über Properties mit Instanzen verknüpft werden.

Festzuhalten ist, dass die T-Box keine konkreten Instanzen enthält, sondern ähnlich wie bei einer Datenbank das Schema der Klassen festgelegt. In der A-Box befinden sich dagegen die konkreten Instanzen der Klassen (Google-Drive-URL, Hans Peter, Timing App Konzept).

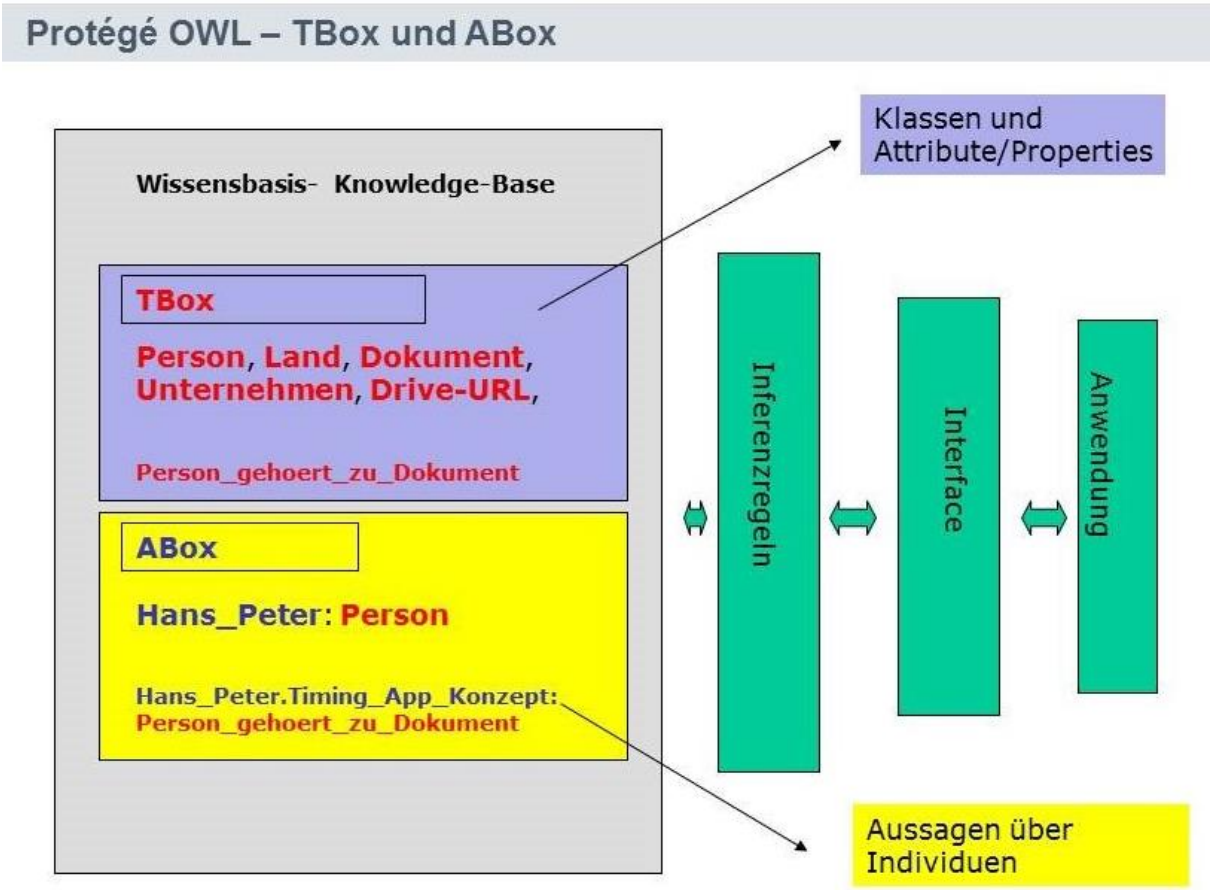


Abbildung 15: Aufbau der Knowledge Base

Bei der Konstruktion unserer Knowledge-Base/Ontology sind wir nach dem Ontology-Vorgehensmodell von On-To-Knowledge vorgegangen. Das On-To-Knowledge Vorgehensmodell wird in Abbildung 16 veranschaulicht.

Abbildung 16: On-To-Knowledge Vorgehensmodell [21, S. 121]

Bei der Konstruktion unserer Knowledge-Base/Ontology haben wir in der T-Box Object Properties, Data Properties sowie Classes verwendet. Classes beschreiben Things und ihre Beziehungen (Subclass) zueinander. Object Properties beschreiben eine Verbindung zwischen zwei Individuals, wie z.B. (**p:John p:arbeitet_für p:Trumpf**).

Data Properties beschreiben eine Verbindung zwischen einem Individual und einem Value (String, Integer), wie z.B. (**p:John p:hasHeight "178.5"^^xsd:float**)**.**

In der A-Box haben wir Annotation Properties, Object Properties, Data properties, Classes, Individuals sowie Annotations verwendet. Annotation Properties verbinden Entitäten (Class, Object Property, Data Property, Annotation Property, Data Type und Individual) mit anderen Entitäten. Individuals enthalten die konkreten Daten (Google-Drive-Link, Instanzen), wie in Abbildung 17 zu sehen ist.

```
<!--
    http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument1
-->
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument1">
    <ab_cloud:ErstellugsDatum rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">01.01.2016 7:55</ab_cloud:ErstellugsDatum>
    <ab_cloud:Version rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</ab_cloud:Version>
    <ab_cloud:DokumentTyp rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dokument</ab_cloud:DokumentTyp>
    <ab_cloud:Status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Technische Realisierung</ab_cloud:Status>
    <ab_cloud:URL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        https://docs.google.com/document/d/1fOFkCM6wb_Ux4za45iQL1sqbVfT1ceO6eTMiZZwiy70
    </ab_cloud:URL>
    <ab_cloud:Autor rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Haruki_Murakami"/>
</owl:NamedIndividual>
<!--
    http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument2
-->
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Testdokument2">
    <ab_cloud:Version rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</ab_cloud:Version>
    <ab_cloud:ErstellugsDatum rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">12.05.2016 13:45 </ab_cloud:ErstellugsDatum>
    <ab_cloud:Status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Anfangsphase</ab_cloud:Status>
    <ab_cloud:DokumentTyp rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dokument</ab_cloud:DokumentTyp>
    <ab_cloud:URL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        https://drive.google.com/file/d/0B1EqTHK_9uoAMDFhZmhicFgwYWc/view?usp=sharing
    </ab_cloud:URL>
    <ab_cloud:Autor rdf:resource="http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Simone_Temperamentos"/>
</owl:NamedIndividual>
<!--
    http://www.semanticweb.org/bjorn/ontologies/2016/11/ab_cloud#Textdokument10
-->
```

Abbildung 17: Beispiel Individuals

4.3.2 Modellierung der Ontologie

Die Ontologie besteht aus den drei OWL-Hauptklassen Geoinformationen, Person und Unternehmen (siehe Abbildung 18). Diesen Hauptklassen wurden mehrere Unterklassen hierarchisch untergeordnet. Den Unterklassen wurden wiederum weitere Unterklassen zugeordnet. Die Hierarchie der Ontologie-Struktur wird in diesem Abschnitt genauer erläutert. Es wird außerdem dargestellt, weshalb diese Klassen relevant für die semantische Repräsentation von Projektdokumenten ist.

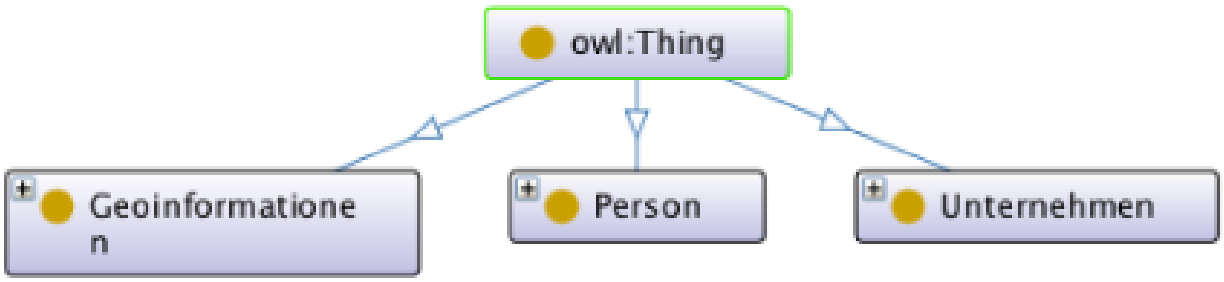


Abbildung 18: OWL-Hauptklassen

Die Klasse Land ist der Klasse Geoinformationen untergeordnet (siehe Abbildung 19). Der Klasse Land ist wiederrum die Klasse Bundesland untergeordnet, da ein Land aus mehreren Bundesländern besteht. Außerdem gibt es eine Beziehung zwischen Bundesland und Land, da ein Bundesland zu einem Land gehört. Ein

Bundeland enthält mehrere Städte, daher ist die Klasse Stadt der Klasse Bundesland untergeordnet. Die Klasse Unternehmen und die Klassen Stadt besitzen eine Relation, da ein Unternehmen einen Standort hat.

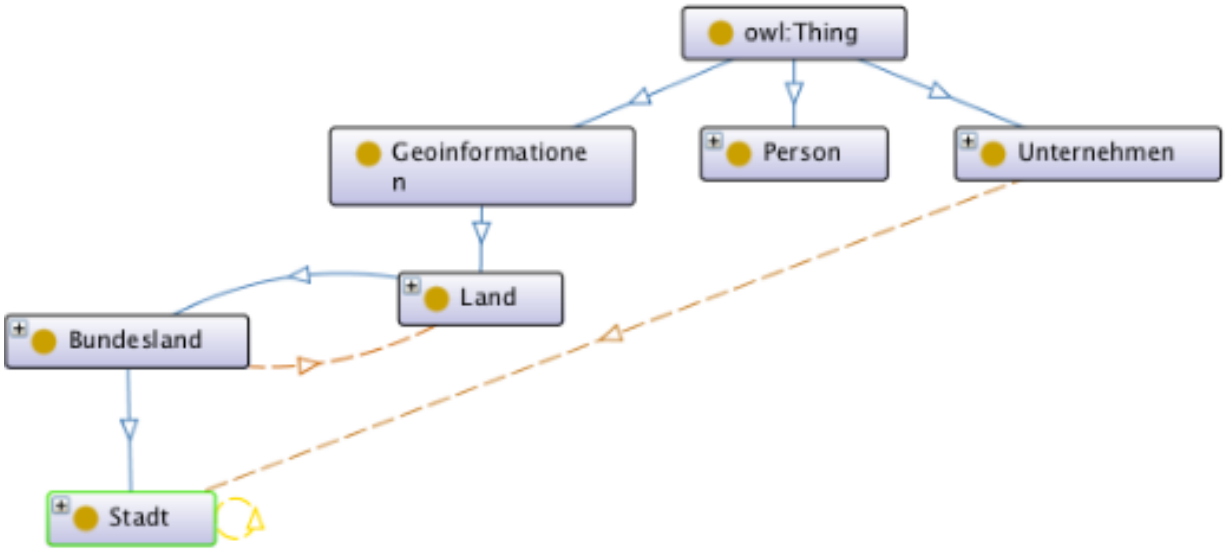


Abbildung 19: OWL-Klasse – Geoinformationen

Der Klasse Person sind die Klassen Kunde und Mitarbeiter zugeordnet (siehe Abbildung 20). Die Klasse Person könnte weiter aufgeteilt werden, jedoch sind ausschließlich der Kunde und der Mitarbeiter relevant für den Prototyp, da in der Regel Kun den und Mitarbeiter an einem Projekt beteiligt sind. Die Klasse Mitarbeiter wurde wiederum aufteilt in interne und externe Mitarbeiter, da es sowohl interne als auch externe Projektmitarbeiter gibt. Außerdem besteht eine Relation zwischen den Klassen Mitarbeiter und Unternehmen, da ein Mitarbeiter in einem Unternehmen angestellt ist.

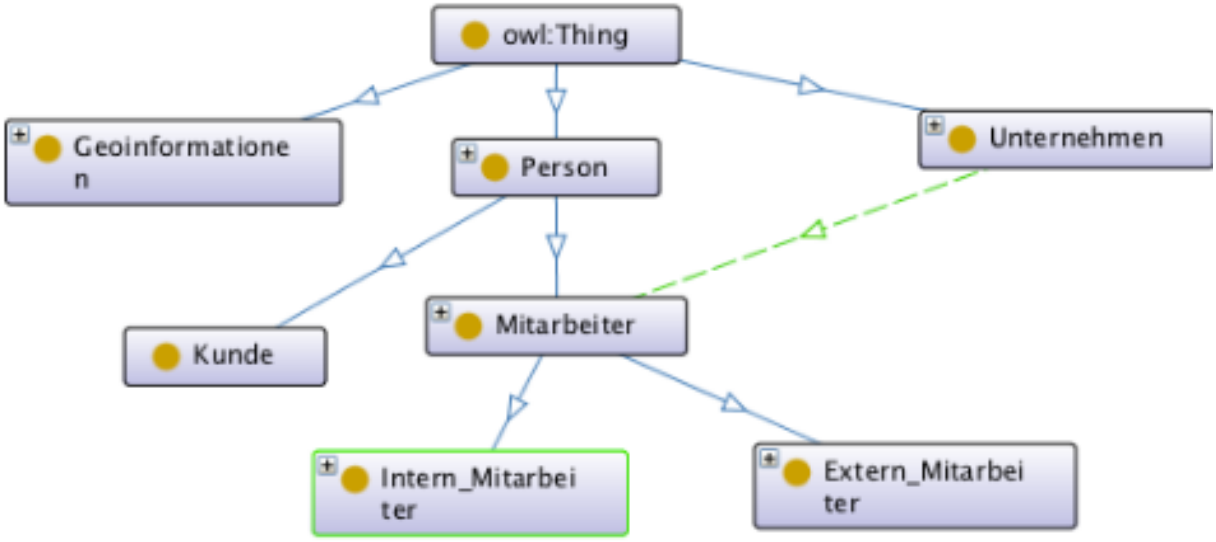


Abbildung 20: OWL-Klasse – Person

Der Klasse Unternehmen sind die Klassen Dokument, Abteilung und Projekt untergeordnet (siehe Abbildung 21). Ein Unternehmen besteht aus mehreren Abteilungen, führt Projekte und hat verschiedene Dokumente. Den drei Unterklassen der Hauptklasse Unternehmen sind weitere Klassen untergeordnet (siehe Abbildung 22).

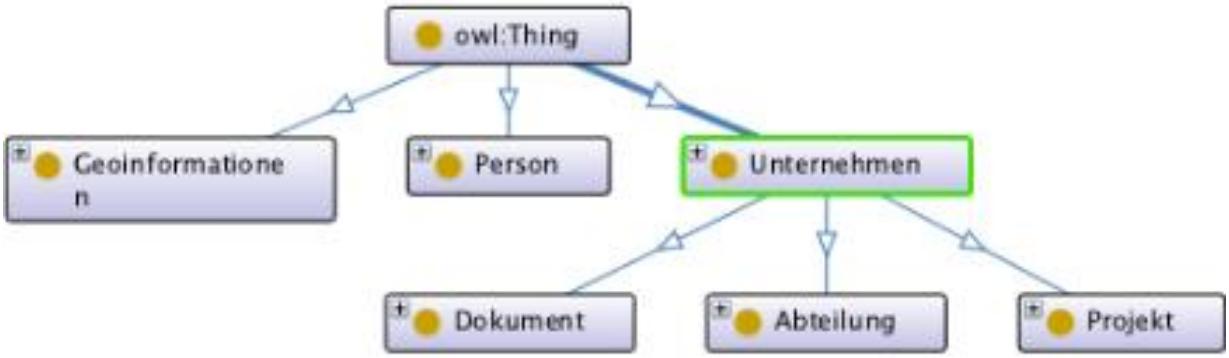


Abbildung 21: OWL-Klasse – Unternehmen

Es gibt verschiedene Abteilungen, wie z.B. Vertrieb, Einkauf, IT, etc. Außerdem gibt es verschiedene Dokumentarten, wie z.B. ein Terminplan, Abschlussbericht, Pflichtenheft, etc. Der Klasse Projekt sind die Klassen Projektrolle und Projekttyp zugeordnet. Die Klasse Projekttyp wird weiter in Forschungs- und Entwicklungsprojekt sowie IT-Projekte unterteilt. Die Unterteilung der Projekttypen ist wichtig, damit der Projekttyp unterschieden werden kann. Außerdem spielt die Projektrolle ebenfalls eine wichtige Rolle für Projekte.

Abbildung 22: Unterklassen der Hauptklasse Unternehmen

4.4 Target Line 2. Semester

Die Machbarkeit und der Technologieumgang ist durch den Prototyp mit den einzelnen Szenarien bewiesen. Dennoch gibt es weitreichende Potentialfelder funktionaler Basis um eine benutzerfreundliche und cloudfähige Applikation realisieren zu können. Folgend haben wir bestimmte Barrieren bzw. Ziele aufgelistet die wir aus Entwicklungsperspektive auf das 2. Semester auslagern. Im Folgenden werden die Ziele kurz aufgeführt:

4.4.1 Google App Script:

- FormularChecker()
  Für diese Funktion für die Prüfung der Formulareingaben des Benutzers. Diese Funktion ist notwendig um Änderungen in den Formularfeldern zu prüfen.

- OWL-Checker()
  Diese Funktion prüft wird beim Klick auf den Button “Speichern” aufgerufen und prüft ob die Metadaten gleich, anders oder gar nicht in der A-Box hinterlegt sind.

- FetchViaUrl()
  Diese Funktion wird nach der OWL-Checker()-Methode aufgerufen. Sollte die OWL-Checker()-Methode als mit dem booleanschem Wert TRUE Antworten so werden die Daten entweder in der OWL-Datei Upgedated oder neu Angelegt.

- Erweiterung um mehr Inhaltstypen.

Denkbar ist auch die manuelle Attributierung auf andere Dateitypen wie zum Beispiel ein Bild oder eine HTML-Datei auszuweiten. Bisher werden nur Dokumente unterstützt.

- Registrierung als Public Google Addon (siehe Abbildung 23)
  Um die Funktion der manuellen Attributierung in der ganzen Google Drive Domäne nutzen zu können ziehen wir in erwägung dies als Public Google Addon zu registrieren.

Abbildung 23: Registrierung Public Google Addon

4.4.2 Google App Engine

- REST-Service
  Ziel ist es den REST-Service auf der Google App Engine auszuführen, dies haben wir im ersten Semester auch schon gemacht, haben allerdings Ausführungsprobleme. Sollten diese nicht gelöst werden, werden wir auf die Google App Engine verzichten und mittels eines Tomcat neben dem Fuseki Server auf der Google Compute Engine laufen lassen. Weiter gilt, bestimmte URL-Endpoints zu definieren, durch die andere Schichten bei uns Daten anfragen können sowie die Methoden Logik und weitere dynamische SPARQL zu schreiben.

Realisierung angedachter Methoden:

- addMetaData() //URL-Fetch Google Apps Script
- checkMetaData() //Triggert By Google Apps Script oder Event-Layer
- getDokcumentByName(),getDokcumentByID(),
- getProjektByTimeIntervall(), etc.

4.4.3 Google Compute Engine

- Fuseki eigene Konfiguration mit persistent Datastore Funktion. Ziel ist es eine eigene Konfigurationsdatei für den FUSEKI-Server zu schreiben, um diesen Systemneutral mit den gleichen Parametern, Abhängigkeiten und Pfaden ausgeführt werden kann. (Fortsetzung im 2. Semester)

5. Fazit und Ausblick

Im Laufe des Wintersemesters 2016/2017 hat unser vierköpfiges Team für das Modul “Technische Grundlagen cloudbasierter Internet Anwendungen“ mehrere lauffähiges semantische Szenarien entwickelt und realisiert.

Diese Szenarien veranschaulichen die Funktionsweise von cloud basierten semantischen Technologien und zeigen diese im realen Einsatz. Konkret veranschaulichen die Szenarien die Abfrage von Ontologie mittels SPARQL. Für die Konzeption unserer Szenarien haben wir unterschiedliche Modellierungsmethoden verwendet, wie zum Beispiel UML-Klassendiagramme oder auch UML-Use-Case-Diagramme.

Nicht nur unsere fachliche Kompetenz sondern auch unsere sozialen Kompetenzen sind in erheblichem Maße gewachsen. Vor allem das Teamwork in unserer Gruppe ist ein wesentlicher Bestandteil unseres Projektes. Ohne die reibungslose Zusammenarbeit untereinander wäre die Umsetzung unserer Szenarien in diesem kurzen Zeitraum nicht möglich gewesen. Durch das Projekt haben wir die Möglichkeit gehabt, unsere theoretischen Kenntnisse in den Bereichen Semantic Web und Semantische Repräsentation an praktische Problemstellungen anzuwenden.

Trotz der zeitintensiven Arbeit an unserem Projekt und die zusätzliche Arbeit an parallel laufenden Projekten haben wir versucht, die Abgabe und die Anforderungen stets zu erfüllen. Dadurch dass wir das erste Mal mit der vorgegebenen Software gearbeitet haben, wie z.B. Semantische Technologien, Protege, Jena oder Apache Fuseki Server sind wir immer wieder auf neue Probleme gestoßen. Diese versuchten wir selbstständig zu lösen oder haben uns Anregungen bei den zuständigen Professoren geholt.

Schlussendlich können wir behaupten, dass wir in diesem Semester nicht nur einiges über das Programmieren von semantischen Anwendungen, sondern auch sehr viel über Teamwork gelernt haben.

Im nächsten Semester (Sommersemester 2017) hat unser vierköpfiges Team für das Modul “Development Lab“ vor, unsere Softwarekomponente (Semantische

Repräsentation) in der Cloud anderen Anwendungen und Softwarekomponenten (Event Processor, Tokenization, etc.) als **Webservice** zur Verfügung stellen. Für die Kommunikation in verteilten Systemen haben sich in den letzten Jahren zunehmend Webservices durchgesetzt.

Über REST-Schnittstellen wird unser Team sämtliche Funktionalitäten der semantischen Systemkomponente in Form von adressierbaren Ressourcen anderen Nutzern sowie Anwendungen bereitstellen. Dies soll mittels Standard-HTTP-Methoden, wie z.B. GET, POST, PUT etc. sowie Jersey realisiert werden.

Ein weiteres Ziel besteht darin, zu versuchen die momentanen Problemursachen (siehe Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.**) zu beheben. Das Google App Script soll hinsichtlich des Funktionsumfangs erweitert werden. Es soll möglich sein, dass die Attributierung automatisiert wird und nicht mehr manuell durchgeführt werden muss. Außerdem sollen alle systemrelevanten Softwarekomponenten in der Google Cloud laufen und Verfügbar sein.

6. Hinweise für Entwickler

Dieser Teil der Dokumentation beinhaltet wichtige Informationen für Personen, die an einer Weiterentwicklung an der Systems-Komponente interessiert sind. Nähere Hinweise zur Funktionalität der Klassen und Methoden können der Java-Doc entnommen werden.

6.1 Eingesetzte Entwicklungs-Werkzeuge

6.1.1 Organisation, Kollaboration und Kommunikation im Team:

- Versionierung: Eclipse Plugin eGit
- Doku: http://www.eclipse.org/egit/?replytocom=14044
- Git Hosting Plattform: github Doku: https://guides.github.com/
- Dokumenten-Management: Google Drive + Google Docs Doku: https://support.google.com/drive/?hl=de#topic=14940
- Projekt-Management: Scrum Doku: https://scrum.com/guide

6.1.2 Modellierung: Ontologie und UML

- OWL-Ontologie
  Doku: http://protege.stanford.edu/
- UML-Plugin Papyrus
  Doku: http://www.eclipse.org/papyrus/
- yEd Graph Editor
  Doku: http://yed.yworks.com/support/manual/index.html

6.1.3 Entwicklung: Hosting, Frameworks, Google Apps Script

- Entwicklungsumgebung - Eclipse Doku: https://eclipse.org/users/
- Google App Engine - Java Runtime Environment Doku: https://cloud.google.com/appengine/docs/java/
- Jena-Framework- Doku: https://jena.apache.org/documentation/
- Fuseki-SPARQL-Server - Doku: https://jena.apache.org/documentation/fuseki2/
- Google Apps Script - Doku: https://developers.google.com/apps-script/reference/document/

Literaturverzeichnis

[1] Neumüller M. (2001): Hypertext Semiotics in the Commercialiazed Internet. Dissertation, Wirtschaftsuniversität Wien, URL: https://core.ac.uk/download/pdf/12236116.pdf?repositoryId=462, [Einsichtnahme: 10.12.2016].

[2] Wikipedia (2015): Repräsentation. URL: https://de.wikipedia.org/wiki/Repräsentation, [Einsichtnahme: 10.12.2016].

[3] Reichenberger, Klaus: Kompendium Semantische Netze- Konzepte, Technologie, Modellierung. Berlin Heidelberg: Springer Verlag, 2010.

[4] Hitzler, P.; Krötzsch, M.; Rudolph, S.; Sure, Y.: Semantic Web: Grundlagen. 1. Aufl. Berlin: Springer Verlag, 2008.

[5] Wikipedia (2016): Unstructured Data. URL: https://en.wikipedia.org/wiki/Unstructured_data, [Einsichtnahme: 12.12.2016].

[6] Pellegrini, T.; Blumauer, A.: Semantic Web. Wege zur vernetzten Wissensgesellschaft. 1. Aufl. Berlin Heidelberg: Springer Verlag, 2006, S.485-524.

[7] W3C (1999): XML Tutorial, URL: http://www.w3schools.com/xml/, [Einsichtnahme: 01.01.2017].

[8] Prud'hommeaux, E.; Seaborne, A. (2008): SPARQL Query Language for RDF. Webseite von W3C, URL: https://www.w3.org/TR/rdf-sparql-query/, [Einsichtnahme: 01.01.2017].

[9] Hitzler, P.; Krötzsch, M.; Rudolph, S.; Sure, Y.: Semantic Web: Grundlagen. 1. Aufl. Berlin: Springer Verlag, 2008.

[10] Huang, J.; Abadi, J./Daniel, Ren, K. (2011): Scalable SPARQL Querying of Large RDF Graphs. Paper, URL: http://cs-www.cs.yale.edu/homes/dna/papers/sw-graph-scale.pdf, [Einsichtnahme: 02.01.2016]

[11] Feigenbaum, Lee (2013): SPARQL 1.1 Protocol, URL: https://www.w3.org/TR/sparql11-protocol/, [Einsichtnahme: 12.01.2017].

[12] Domingue, J., Fensel, D.; Hendler, J.: *Handbook of Semantic Web Technologies*. 1. Aufl. USA: Springer Science & Business Media, 2011.

[13] Fensel, D.; Facca, F. (2008): Semantic Web, URL: http://teaching-wiki.sti2.at/uploads/c/c2/SW-09-Web-scale_Reasoning.pdf, [Einsichtnahme: 16.01.2017].

[14] Kifer, M.; Boley, H. (2009): RIF Overview W3C Working Draft 1 October 2009, URL: https://www.w3.org/TR/2009/WD-rif-overview-20091001/,

[Einsichtnahme: 16.01.2017].

[15] Studer, R., Grimm, S., Abecker, A. (2006): Semantic Web Services: Concepts, Technologies, and Applications, URL: http://semantisches-web.net/technologien/
beweise-vertrauen-digitale-signaturen/, [Einsichtnahme: 16.01.2017, S.136]

[16] Reginald, Ferber (2004): Information Retrieval, URL: http://information-retrieval.de/irb/ir.part_4.chapter_2.section_4.subdiv1_2.html,
[Einsichtnahme: 17.01.2017]

[17] Technische Universität Chemnitz (2008): Semantic Web und Trust. Hauptseminar Web Enigneering, URL: http://wissen.werner-welt.de/semantic_web_trust.html, [Einsichtnahme: 17.01.2017]

[18] Serge, Linckels (2014): Semantic Web – OWL. Präsentation, URL: http://www.slideshare.net/SergeLinckels/semantic-web-overview,
[Einsichtnahme: 17.01.2017, S. 2]

[19] W3C (2012): Prime3, URL: https://www.w3.org/2005/rules/wiki/Primer, [Einsichtnahme: 17.01.2017]

[20] Pellegrini, T.; Blumauer, A.: Semantic Web. Wege zur vernetzten Wissensgesellschaft. 1. Aufl. Berlin Heidelberg: Springer Verlag, 2006.

[21] Sure, York; Staab, Steffen; Studer, Rudi (2004): On-To-Knowledge Methodology (OTKM). Handbook on Ontologies, Berlin Heidelberg: Springer Verlag.