Pedro Alves Zipf
6194978
Informatik (BA)
SoSe 2019
s7333867@stud.uni-frankfurt.de

Bachelorarbeit

Geotagging und Ontologie-basierte Visualisierung für das TextImaging

Pedro Alves Zipf

Abgabedatum: 16.09.2019

Text Technology Lab
Goethe Universität Frankfurt am Main
Prof. Dr. Alexander Mehler

Erklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

_____

Ort, Datum

_____

Unterschrift

Abstract

Zielsetzung dieser Arbeit ist es Nutzern, ohne Programmierkenntnisse oder Fachwissen im Bereich der Informatik, Zugang zu der automatischen Verarbeitung von Texten zu gewährleisten. Speziell soll es um Geotagging, also das Referenzieren verschiedener Objekte auf einer Karte, gehen. Als Basis soll ein ontologisches Modell dienen, mit Hilfe dessen Struktur die Objekte in Klassen eingeteilt werden. Zur Verarbeitung des Textes werden Natural Language Processing Werkzeuge verwendet.

Natural Language Processing beschreibt Methoden zur maschinellen Verarbeitung natürlicher Sprache. Sie ermöglichen es, die in Texten enthaltenen unstrukturierten Informationen in eine strukturierte Form zu bringen. Die so erhaltenen Informationen können für weitere maschinelle Verarbeitungsschritte verwendet oder einem Nutzer direkt bereitgestellt werden. Sollten sie direkt bereitgestellt werden, ist es ausschlaggebend, sie in einer Form zu präsentieren, die auch ohne Fachkenntnisse oder Vorwissen verständlich ist.

Im Bereich der Geographie wird oft der Ansatz befolgt, die erhaltenen Informationen auf Basis verschiedener Karten, also visuell zu verarbeiten. Visualisierungen dienen hierbei der Veranschaulichung von Informationen. Durch sie werden die relevanten Aspekte dem Nutzer verdeutlicht und so die Komplexität der Informationen reduziert.

Es bietet sich also an, die durch das Natural Language Processing gesammelten Informationen in Form einer Visualisierung für den Nutzer zugänglich zu machen. Im Rahmen dieser Arbeit über Geotagging und Ontologie-basierte Visualisierung für das TextImaging wird ein Tool entwickelt, das diese Brücke schlägt. Die Texte werden auf einer Karte visualisiert und bieten so eine Möglichkeit, beschriebene geographische Zusammenhänge auf einen Blick zu erfassen. Durch die Kombination der Visualisierung auf einer Karte und der Markierung der entsprechenden Entitäten im Text kann eine zuverlässige und nutzerfreundliche Visualisierung erzeugt werden. Bei einer abschließenden Evaluation hat sich gezeigt das mit dem Tool der Zeitaufwand und die Anzahl der fehlerhaften Annotationen reduziert werden konnte.

Die von dem Tool gebotenen Funktionen machen dieses auch für weiterführende Arbeiten interessant. Eine Möglichkeit ist die entwickelten Annotatoren zu verwenden um ein ontology matching auf Basis bestimmter Texte auszuführen. Im Bereich der Visualisierung bieten sich Projekte wie die Visualisierung historischer Texte auf Basis automatisch ermittelter, zeitgerechter Karten an.

Danksagung

An dieser Stelle möchte ich mich bei all jenen herzlich bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben. Mein besonderer Dank gilt Herrn Prof. Dr. Alexander Mehler und Herrn Giuseppe Abrami von der Goethe-Universität Frankfurt für ihre Betreuung und technische Unterstützung. Ebenso danke ich herzlich meinen Freunden, Kommilitonen und meiner Familie, die mir während dieser Arbeit und während meines gesamten Studiums zur Seite standen.

Inhaltsverzeichnis

1. Einleitung **8**
- 1.1. Motivation . . . 8
- 1.2. Aufgabenstellung . . . 9
- 1.3. Methodik . . . 9
- 1.4. Struktur der Arbeit . . . 9

2. Verwendete Werkzeuge und Modelle **10**
- 2.1. Modelle . . . 10
  - 2.1.1. UIMA . . . 10
  - 2.1.2. TextImager . . . 10
  - 2.1.3. WikiData . . . 11
  - 2.1.4. Ontologie . . . 11
- 2.2. Werkzeuge . . . 13
  - 2.2.1. Natural Language Processing . . . 13
  - 2.2.2. Apache Jena . . . 16
  - 2.2.3. TagMeAPIAnnotator . . . 16
  - 2.2.4. GeoViz Webapp . . . 16
  - 2.2.5. Nominatim . . . 16

3. Algorithmische Konzeption **17**
- 3.1. Datenabfragen . . . 17
- 3.2. Ontologisches Modell . . . 18
- 3.3. UIMA Datentypen . . . 18
  - 3.3.1. RelationContext . . . 18
  - 3.3.2. IndividualVis . . . 19

4. FrontEnd **20**
- 4.1. Visualisierung geographischer Informationen . . . 21
  - 4.1.1. Popups . . . 21
  - 4.1.2. Erkennung von Überschneidungen . . . 22
- 4.2. Visualisierung des Textes . . . 22
  - 4.2.1. Scroll Panel . . . 22
  - 4.2.2. Markierung von Wörtern . . . 23

5. BackEnd **24**
- 5.1. Ontologisches Modell . . . 24
  - 5.1.1. OntologyStore . . . 24
- 5.2. Annotatorpipeline . . . 25
  - 5.2.1. OntologyOptimizer . . . 26

5.2.2. RelationAnnotator 26

6. Evaluation **28**
6.1. Aufgabenstellung 28
6.2. Durchführung der Befragung 28
6.3. Auswertung 29
6.4. Ergebnisse 29
6.4.1. Annotation 29
6.4.2. UMUX-Fragebogen 31
6.4.3. Schlussfolgerung 31

7. Fazit und Ausblick **33**
7.1. Fazit 33
7.2. Anwendungsmöglichkeiten 34
7.2.1. TextImager 34
7.2.2. TextAnnotator 34
7.3. Desiderata (Offene Probleme) 35
7.3.1. Ontologisches Modell 35
7.3.2. TagMeAnnotator 35
7.4. Future Work 35
7.4.1. Optimierung des geocodings 35
7.4.2. Bereitstellung zusätzlicher Operationen 36
7.4.3. Klassifikation von Texten 36
7.4.4. Ontology matching 36
7.4.5. Erweiterung der visualisierten Struktur 36
7.4.6. Erweiterung der Querverweise 37
7.4.7. Visualisierung historischer Texte 37

A. Abbildungen **42**

B. README **43**

Abbildungsverzeichnis

2.1. Auszug: Environmental Ontology . . . 11
2.2. LanguageToolSegmenter . . . 13
2.3. LanguageToolLemmatizer . . . 13
2.4. StanfordPOSTagger . . . 14
2.5. NamedEntityRecognizer . . . 14
2.6. DependencyParser . . . 14
2.7. ParagraphSplitter . . . 15
2.8. TagMe Annotator . . . 15
2.9. WikiDataHyponyms . . . 15

3.1. SPARQL Query . . . 17
3.2. Typesystem Struktur . . . 19

4.1. GeoViz Tool . . . 20
4.3. Popup . . . 21
4.4. Highlighting Map . . . 22
4.5. HighlightingWord . . . 23

5.1. Annotation Pipeline . . . 25

6.2. Auswertung: UMUX Fragebogen . . . 31

A.1. SaveButoon & NamePrompt . . . 42
A.2. Scroll Panel . . . 42
A.3. Scroll Panel for 1 object . . . 43

1. Einleitung

1.1. Motivation

Diese Arbeit folgt dem Vorbild des Fachinformationsdienstes Biodiversitätsforschung (Weiland u. a., 2018) um einen Beitrag zur um einen Beitrag zur Geovisualisierung von Biodiversitätsliteratur zu leisten. Die geographische Visualisierung eines Textes ist hierbei aus folgenden Gründen sinnvoll:

Visualisierungen (Fekete u. a., 2008) sind eine Methode, um Informationen für einen Beobachter intuitiv darzustellen. Visualisierte Informationen sind für die meisten Menschen leichter verständlich und können auch ohne Fachwissen interpretiert werden. Sie fungieren als eine Art Auslagerung für das Gehirn, sodass mehr Kapazität für die Auswertung dieser Informationen bereitgestellt werden kann. Ein passendes Beispiel findet sich in der Mathematik: Eine auf Papier einfach zu bewerkstelligende Division kann ohne dieses visuelle Hilfsmittel für viele Menschen – vor allem Personen, die in ihrem Alltag wenig mit mathematischen Gleichungen zu tun haben – schwer oder gar nicht zu bewerkstelligen sein. Um die Visualisierung eines Textes zu ermöglichen, müssen zuerst Informationen aus dem Text gelesen werden.

Bei der Extraktion der Informationen geht es darum, die in Texten verwendete natürliche Sprache so zu verarbeiten, dass relevante Daten von einem Computer erkannt und gespeichert werden können. Sowohl die Verwendung entsprechender Tools als auch die Auswertung der ermittelten Daten setzen Kenntnisse in einem oder mehreren Bereichen der Informatik voraus. Deswegen soll in dieser Arbeit ein Werkzeug geschaffen werden, das es allen Nutzern erlaubt, geographische Daten aus einem beliebigen Text zu extrahieren, ohne Vorkenntnisse vorauszusetzen.

Das Projekt folgt hierbei den Prinzipien, auf denen der in Text Technology Lab entwickelte TextImager aufbaut. Jedem Nutzer sollen Ergebnisse aus Natural Language Processing Tools zugänglich gemacht werden, ohne Fachkenntnisse oder Vorwissen zu erfordern. Der im Rahmen dieser Arbeit entwickelte TextToMap Annotator bietet zwei Funktionalitäten, um die genannten Anforderungen zu erfüllen. Er kann einen Text einlesen und alle NLP-Prozesse ausführen, die für eine Visualisierung notwendig sind. Anschließend wird der annotierte Text auf einer Karte visuell dargestellt – das bietet dem Nutzer verschiedene Interaktionsmöglichkeiten.

1.2. Aufgabenstellung

Das Ziel dieser Arbeit ist es, ein Programm zu entwickeln, das die strukturierte Visualisierung geographischer Informationen aus einem Text auf einer Karte ermöglicht. Hierbei soll zur Strukturierung der Informationen ein ontologisches Modell verwendet werden. Als Basis der Visualisierung soll das im TTLab entwickelte GeoViz Tool dienen. Das Programm soll außerdem die Voraussetzungen erfüllen, um in den TextImager (s. Abschnitt 2.1.2) eingebunden werden zu können.

1.3. Methodik

Die Entwicklung des Tools lässt sich grob in drei Abschnitte teilen. Zuerst wurde ein Modell entwickelten in dem der Ablauf des Programms definiert ist. Anschließend steht die Erarbeitung eines UIMA-Typsystems (s. Abschnitt 3.3). Hier wurde festgelegt welche Daten für die Visualisierung benötigt werden und damit auch wie diese aussehen. Der letzte Schritt beinhaltet das Programmieren. Also die Umsetzung des in Schritt eins entwickelten Modells auf Basis des in Schritt zwei erarbeiteten Typsystems. Entwickelt wurde hierbei in zwei Programmiersprachen. Für die Annotation des Textes wurde die Programmiersprache Java verwendet. Die Visualisierung in einer Webapplikation ist in Javascript realisiert.

1.4. Struktur der Arbeit

Die vorliegende Arbeit ist folgendermaßen aufgebaut:
Kapitel eins beschreibt das Ziel dieses Projektes und warum die Visualisierung geographischer Informationen sinnvoll sein kann. In Kapitel zwei wird auf verwendete Werkzeuge und Modelle eingegangen. Diese schaffen die Grundlage, auf der das Projekt aufbaut. Anschließend wird in Kapitel drei auf algorithmische Konzepte Bezug genommen. Sie beinhalten den Aufbau der Datenabfragen und die Wahl der verwendeten ontologischen Modelle. Außerdem wird auf die erzeugten UIMA-Datentypen eingegangen und ihre Funktion im Zusammenhang des gesamten Projekts erläutert. In Kapitel vier wird das FrontEnd vorgestellt. Seine Funktionalitäten werden erläutert und es wird auf die verwendeten Daten ausgewichen. In Kapitel fünf werden die für das BackEnd entwickelten Klassen beschrieben. Diese finden vor und in der Annotation der für die Visualisierung benötigten Daten Anwendung. In Kapitel sechs folgen die Beschreibung und Auswertung durch Evaluationsergebnisse. Abschließend finden sich in Kapitel sieben das Fazit der Arbeit und ein Ausblick auf verschiedene Anwendungsmöglichkeiten.

2. Verwendete Werkzeuge und Modelle

Zur Realisierung der in der Einleitung formulierten Anforderungen werden verschiedene texttechnologische Werkzeuge und Modelle benötigt. Diesbezüglich gibt es bereits Vorarbeiten, die die in TextToMap Annotator benötigten Funktionalitäten erfüllen können. In diesem Kapitel wird auf diese Komponenten eingegangen, um ihren Zweck und Einsatz im Gesamtprojekt zu erläutern. Diese sind aufgeteilt in verwendete Modelle und Werkzeuge. In dem Abschnitt Modelle wird auf Komponenten eingegangen, die eine Struktur, die in dem Projekt verwendet wird, bereitstellen oder generieren. Der TextImager wird hier wie ein Modell behandelt, da er die für das Projekt benötigte Datenstruktur vorgibt und die einzelnen Tools des TextImager nicht getrennt verwendet werden. Anschließend wird auf verwendete Werkzeuge eingegangen. Hiermit sind Komponenten gemeint, die eine Funktionalität für die Software bereitstellen. Inbegriffen sind hier sowohl importierte Annotatoren als auch Frameworks und Online-API-Dienste.

2.1. Modelle

2.1.1. UIMA

Das UIMA (Unstructured Information Management Architecture) Framework ist eine Architektur zur Verwaltung unstrukturierter Informationen. Verwendet wird es in Data-Mining-Anwendungen, d. h. Anwendungen, die Informationen aus verschiedenen Quellen extrahieren. In dem vorliegenden Projekt wird das UIMA Framework herangezogen, um die aus dem Text gewonnenen Informationen in der für die Weiterverarbeitung nötige Struktur abzuspeichern.
Genauer werden im UIMA Framework Datentypen erstellt, deren Attribute darauf ausgelegt sind, Informationen zu speichern, die für die Visualisierung sinnvoll sind. Diese UIMA-Datentypen werden als CAS-Datei gespeichert und anschließend über eine Formatierung in den xml.-Datentyp für die Webapplikation zugänglich gemacht. Zum speichern und

2.1.2. TextImager

TextImager (Hemati, Uslu und Mehler, 2016) ist ein auf UIMA basierendes Framework zur Bereitstellung verschiedener Natural Language Processing (s. 2.2.1) - und Visualisierungstools. Es hat zum Ziel, ein NLP Framework zu bieten, das auch von Nutzern ohne Programmierkenntnisse oder fachliches Vorwissen benutzt werden kann. Unterstützt wird das durch den Fokus auf der Verständlichkeit der Visualisierungen und eine nutzerfreundliches User Interface.
Das hier behandelte Projekt wird auf Basis von Klassen, die auf dem UIMA Framework aufbauen, konzipiert und über eine Webapplikation visualisiert. Das bietet die Möglichkeit, das Projekt in TextImager zu intergrieren.

2.1.3. WikiData

WikiData ist eine frei zugängliche kollaborative Wissensdatenbank (Vrandečić und Krötzsch, 2014). Sie hat zum Ziel, die Strukturierung und das Management der in ihren Schwesterprojekten (Wikipedia, Wiktionary, et al.) enthaltenen Daten zu optimieren, und wird auch als das Wikipedia der Daten bezeichnet.

In dem vorliegenden Projekt dient die Datenbank als Referenz für die Erweiterung des ontologischen Modells. Hinter WikiData steht die WikiProject Ontology. Diese definiert Klassen als Entitäten, die mit den Eigenschaften Instance of und Subclass of identifiziert werden können. Diese Logik wird genutzt, um Objekte, die die Eigenschaft Instance of aufweisen, als individual zu erkennen und zu speichern. Die Eigenschaft Subclass of markiert Überklassen der entsprechenden Klasse. Zu beachten ist hierbei, dass nach der verwendeten Logik jedes Objekt, das als individual in das ontologische Modell importiert wird, keine Klasse sein kann und somit auch keine Unter- und Überklassen besitzt.

2.1.4. Ontologie

Der Ursprung des Begriffes Ontologie reicht bis in das 15 Jahrhundert zurück und er bezeichnet in der Philosophie die ‚Lehre vom Sein'. Sie befasst sich mit der Strukturierung aller existierenden Gegenstände.

Bei der Ontologie in der Informatik handelt es sich um ein Konzept von Beziehungen zwischen einer Gruppe verschiedener Entitäten. Es wird versucht, eine Struktur zu erzeugen, die die Relationen dieser Entitäten bezogen auf einem bestimmten Themenbereich wiedergibt.

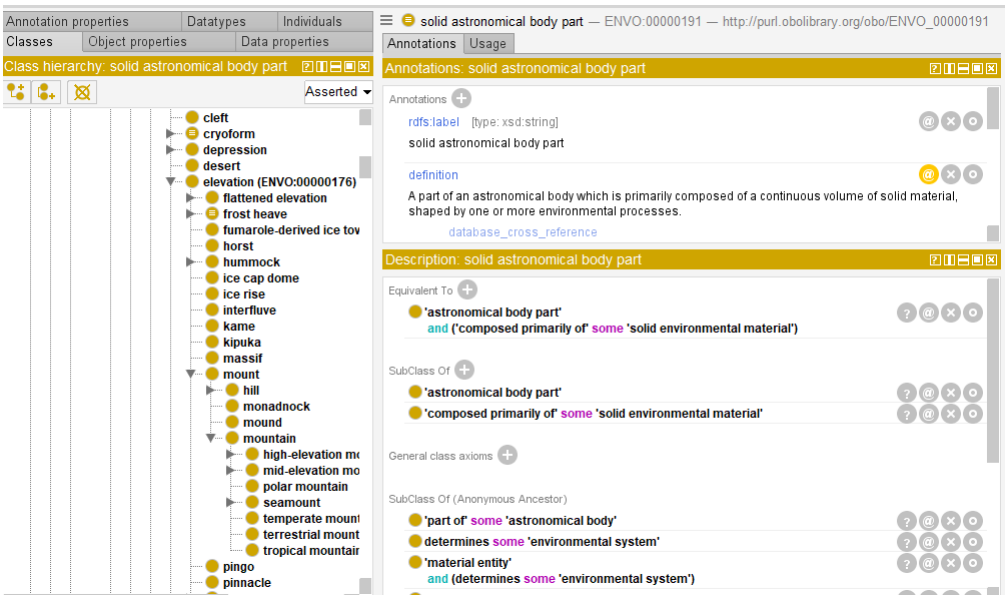


Abbildung 2.1.: Auszug: Environmental Ontology

Environmental ontology

Die ENVO (Environmental Ontology) (Buttigieg u. a., 2013) bildet die Basis für das in dem Projekt verwendete ontologische Modell. Sie ist ein im OWL-Format entwickeltes Modell zur Beschreibung der Umwelt. Darin beinhaltet sind verschiedene Umweltprozesse und Objekte sowie Ökosysteme. Da das gegebene Projekt die Visualisierung verschiedener Named Entity in einem Text zum Ziel hat, sind in erster Linie die geographischen Features relevant, die diese Ontologie bietet (s.2.1).

Geographical entity ontology

Bei der GEO (Geographical Entity Ontology) handelt es sich um eine in OWL2 implementierte Ontologie, in der geographische Entitäten strukturiert sind. Sie beinhaltet sowohl Nationen und Staaten als auch alle Arten geographischer Objekte. Beispiele hierfür sind verschiedene Arten von Gewässern und Landformationen. Die im Folgenden aufgelisteten ontologischen Modelle sind die Bereiche der GEO, die für das vorliegende Projekt verwendet werden.

basic-geography

In dem Bereich Basic Geography befinden sich Klassen zur Strukturierung materieller, geographischer Objekte. Er dient zur Erweiterung der in der ENVO bereits enthaltenen Landformationen. Ein Beispiel hierfür sind Wüsten, Gebirge oder Wälder.

nation-geography

Nation Geography dient der Strukturierung verschiedener Nationen. Unter Nationen werden in der Ontologie geographische Regionen verstanden, die einem ‚Souvereign State' zuzuordnen sind.

body-of-water

Bodies of Water sind Klassen zur Strukturierung verschiedener Gewässer. Sie umfassen Klassen wie Flüsse, Seen und Meere.

2.2. Werkzeuge

2.2.1. Natural Language Processing

„Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.“
Liddy (2001)

Mit ‚Natural Language Processing‘ sind nach diesem Zitat Verfahren in der Textverarbeitung gemeint, die versuchen, möglichst menschenähnlich Informationen aus einem Text zu gewinnen. Im Folgenden werden die für das vorliegende Projekt relevanten NLP-Verfahren beschrieben und veranschaulicht.

LanguageToolSegmenter

Der LanguageToolSegmenter (The DKPro Core Team, 2019) erkennt Sätze und Token in einem Text. Diese bilden die Grundlage für weitere Annotationen.

Abbildung 2.2.: LanguageToolSegmenter

LanguageToolLemmatizer

Der LanguageToolLemmatizer (The DKPro Core Team, 2019) sucht für jedes Token in den Language Tool Lexica nach dem meist verwendeten Lemma und fügt dieses der JCas Datei hinzu. Ist kein Lemma vorhanden wird das Token als neues Lemma gesetzt.

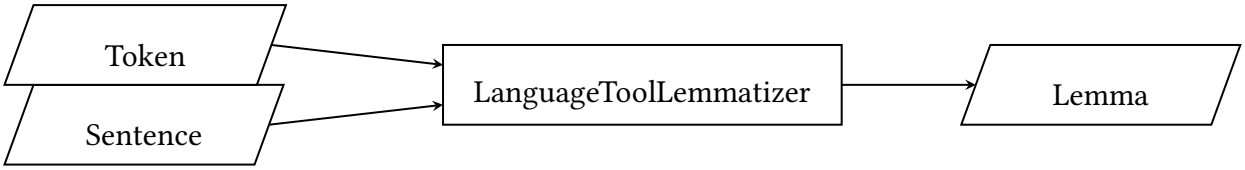


Abbildung 2.3.: LanguageToolLemmatizer

StanfordPOSTagger

Der StanfordPOSTagger (Manning u. a., 2014) ermittelt unter Zuhilfenahme der generierten Sätze für jedes Token die Part-of-Speech, also die Wortart. .

Abbildung 2.4.: StanfordPOSTagger

Stanford NamedEntityRecognizer

Der Stanford NamedEntityRecognizer (The DKPro Core Team, 2019) ermittelt mit Hilfe der Sätze und Token alle Named Entity die in dem gegeben Textcorpora enthalten sind.

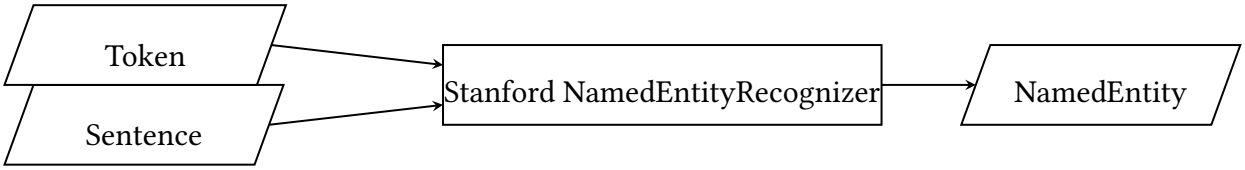


Abbildung 2.5.: NamedEntityRecognizer

CoreNlp DependencyParser

Der CoreNlpDepenedency Parser (Manning u. a., 2014) benutzt die generierten Sätze, Token und POS um die grammatikalische Struktur des Satzes zu repräsentieren.

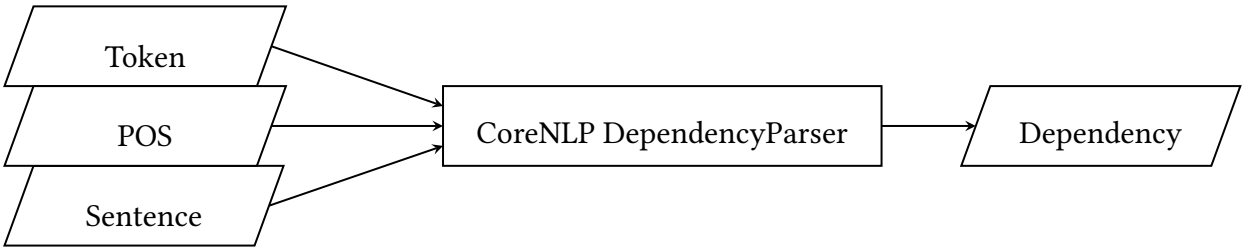


Abbildung 2.6.: DependencyParser

ParagraphSplitter

Der ParagraphSplitter (Manning u. a., 2014) teilt den gegebenen Textkorpus in Paragraphe auf. Hierbei sind einer oder mehr Zeilenumbrüche als Grenze für einen Paragraphen definiert.

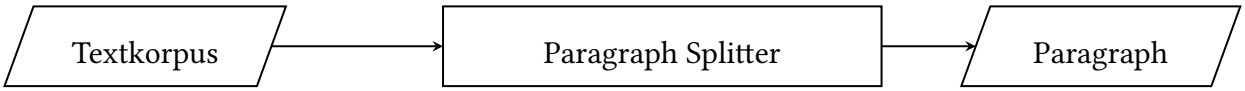


Abbildung 2.7.: ParagraphSplitter

TagMe Annotator

TagMe Annotator (Ferragina und Scaiella, 2012) ist ein vom A^3 Lab entwickelter Entity Linker. Das bedeutet, er findet Tags in einem Text und verlinkt diese mit ihrem entsprechenden Wikipedia-Artikel. Als Tags werden hierbei alle Objekte bezeichnet, die Nomen sind und denen ein entsprechender Wikipedia-Artikel zugewiesen werden kann. Die Genauigkeit dieser Links kann angepasst werden, um die Anzahl der erhaltenen Treffer zu optimieren

Abbildung 2.8.: TagMe Annotator

WikidataHyponyms

WikiDataHyponyms ist ein Annotator, der unter Verwendung der in TagMe Annotator ermittelten Wikipedia-Einträge weitere WikiData-Einträge generiert, die mit der ermittelten Wikipedia-Seite in Verbindung stehen. Jedem dieser Einträge wird eine Tiefe zugewiesen, um aufsteigend von -1 die Überklassen des Objekts zu repräsentieren.

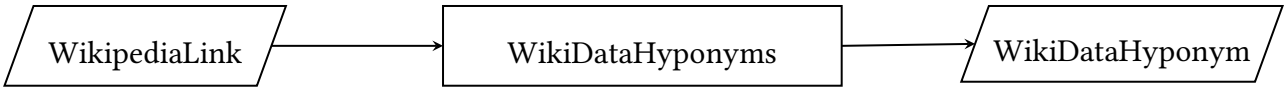


Abbildung 2.9.: WikiDataHyponyms

2.2.2. Apache Jena

Apache Jena OntModel

Apache Jena OntModel ist eine Klasse zur Speicherung eines Modells, das ontologische Daten enthält. Es bietet einfache Möglichkeiten, ontologische Modelle zu laden und Abfragen an diese zu senden. Für das vorliegende Projekt ist die Unterstützung von Daten im OWL-Format relevant, da so über eine PURL alle gewünschten Modelle geladen werden können.

Apache Jena TDB

Apache Jena TDB ist eine Komponente von Jena zur Speicherung und Abfrage von RDF-Modellen auf einem einzelnen Host. Dabei sollte nur dieser Zugriffe auf das gespeicherte Modell tätigen, da sonst Daten korrumpiert werden können.

Das RDF-Modell kann in verschiedenen Formaten gespeichert werden. Das im Projekt verwendete Speicherformat entspricht dem für OWL definierten Standard.

2.2.3. TagMeAPIAnnotator

Der TagME API Annotator ist eine von Wahed Hemati geschriebene Klasse, die dazu dient, den Zugriff auf TagMe Annotator API zu organisieren. In dieser Klasse werden die für die API-Abfrage nötigen Informationen ermittelt. Anschließend wird eine Abfrage an den von D4Science gehosteten Dienst gesendet. Die erhaltenen Informationen werden wie auch bei den anderen Annotatoren im CAS-Format gespeichert, um für nachfolgende Annotatoren zugänglich gemacht zu werden.

2.2.4. GeoViz Webapp

Die GeoViz-Webapp ist eine unter Aufsicht von Tolga Uslu in TTLab entwickelte Sammlung von Softwarebausteinen zur Visualisierung von Informationen auf Basis einer OpenStreetMap Karte. Enthalten sind hier grundlegende Bibliotheken wie MapBox und Leaflet, die Tools zur Visualisierung geographischer Informationen bieten.

Außerdem findet sich hier ein HTML Framework zur Visualisierung von Text und Karte. Dieses besteht aus einer Navigationsleiste, einer Spalte für das Textdokument und der Karte, auf der die Visualisierungen angezeigt werden.

2.2.5. Nominatim

Nominatim (Clemens, 2015) ist eine von OpenStreetMap bereitgestellte API zur Suche von OpenStreetMap-Daten. Für die Suche werden hier jeweils Labels oder Adressen verwendet die bestimmten OpenStreetMap-Daten zugeordnet sind. In der Suche kann angegeben werden, welches Format die Ausgabe haben soll. In dem vorliegenden Projekt wird das geoJson-Format verwendet, da es von der für die Visualisierung verwendeten Javascript-Bibliothek unterstützt wird.

3. Algorithmische Konzeption

In diesem Kapitel der Arbeit wird auf verschiedene Konzepte eingegangen, die für die Realisierung des Projekts von Bedeutung sind. Sie umfassen die benötigten Datenabfragen, die verwendeten ontologischen Modelle und UIMA-Datentypen zum Speichern der Informationen.

3.1. Datenabfragen

Für die Datenabfrage wird SPARQL (SPARQL Protocol and RDF Query Language) verwendet. Das ermöglicht es, Abfragen an WikiData und an das lokal gespeicherte ontologische Modell zu senden. Die Daten sind in Triple gespeichert, sodass über ein Matching der Objekte in Triple eine Suche ausgeführt wird. (s.3.1)

```
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
        "SELECT DISTINCT ?itemLabel ?item ?supItem ?supItemLabel ?geonamesID WHERE \n" +
        "{ \n" +
        "  ?item ?p \""+ label.trim() +"\"@en . \n" +
        "  ?item rdfs:label ?itemLabel .\n" +
        "  ?item wdt:P279 ?supItem .\n" +
        "  ?supItem rdfs:label ?supItemLabel .\n" +
        "  FILTER (lang(?supItemLabel)=\"en\") .\n" +
        "  FILTER (lang(?itemLabel)=\"en\") .\n" +
        "}" +
        "Limit 5";
```

Abbildung 3.1.: SPARQL Query

In dem vorliegenden Projekt werden zwei Arten von SPARQL-Abfragen benötigt: eine Abfrage an einen WikiData Server und eine Abfrage an das lokal gespeicherte ontologische Modell. Der Aufbau dieser Abfragen ist identisch, wobei bei WikiData-Abfragen die von WikiData selbst definierten Properties verwendet werden können. Für Abfragen des ontologischen Modells werden ausschließlich für das RDF-Format zulässige Abfragen verwendet.

3.2. Ontologisches Modell

Bei der Auswahl der Modelle lag besonderer Wert auf der Struktur der geographischen Objekte, die in diesen enthalten sind. Da das ontologische Modell die Visualisierung von Objekten auf einer Karte unterstützen soll, muss es biologische Informationen zu Umwelt, Ökosystemen und Lebensräumen enthalten. Das Hauptaugenmerk liegt in diesem Projekt allerdings auf der Bezeichnung verschiedener Gewässer und Landformationen.
Untersucht wurden folgende Ontologien:

1. EDAM: Die EDAM enthält etablierte Konzepte der Bioinformatik wie Datentypen und Prozesse. In dieser Ontologie fehlt der geographische Ansatz, den das Programm benötigt.

2. MEO: Die Metagenome and Microbes Environmental Ontology enthält Lebensräume verschiedener Organismen. Diese Ontologie bietet einen guten Ausgangspunkt für das Projekt, da viele Lebensräume auch geographischen Landformen entsprechen (Vulkan, Sumpf, Wüste etc.). Sie wird allerdings nicht verwendet, da auch die Environmental Ontology diese Aspekte abdecken kann.

3. ENVO: Environmental Ontology wird für das vorliegende Projekt verwendet und in Abschnitt 2.1.4 genauer erläutert.

4. GEO: Die Geographical Entity Ontology wird verwendet, um die in der ENVO nicht vorhandenen Strukturen für von Menschen geschaffene Nationen abzudecken. Außerdem enthält sie Ergänzungen zu Gewässern, die in der ENVO nicht vorhanden sind. Aus der GEO-Ontologie werden nur einzelne Untermodelle importiert (siehe Abschnitt 2.1.4).

3.3. UIMA Datentypen

Als Basis für eine modellbasierte Programmierung verwendet der Annotator ein UIMA-Typsystem (s. 2.1.1). Das ermöglicht einen flexiblen Wechsel der Annotatoren, solange die Datentypen entsprechend eingehalten werden. Der UIMA-Datentyp ist außerdem eine Voraussetzung, die das Programm erfüllen sollte, um eine Integration in den TextImager vorzunehmen (s. 2.1.2). In dem vorliegenden Projekt wird das Typsystem um zwei Datentypen erweitert, die für die Visualisierung optimale Attribute aufweisen. Abbildung 3.2 stellt das Typsystem in einem UML-Diagramm dar.

3.3.1. RelationContext

RelationContext ist ein Datentyp zur Strukturierung von Informationen auf Basis eines ontologischen Modells. Der Datentyp beinhaltet eine ID zu dem Hyponym, dessen Kontext er bilden soll, ein Label und eine Relation zu der Klasse innerhalb der Ontologie, zu der das Hyponym gehört. Mit Hilfe des RelationContext kann während der Visualisierung eine Struktur aufgebaut werden, die der des ontologischen Modells entspricht.

3.3.2. IndividualVis

Der IndividualVis-Datentyp dient zur Speicherung von Informationen, die zur Visualisierung benötigt werden. Jedes IndividualVis wird per Referenz-ID einem Hyponym zugewiesen und speichert für dieses eine geoJson sowie ein Label und die WikiData ID. Für jedes dieser Objekte wird in der Webapplikation ein Layer erstellt. Jedes IndividualVis-Objekt entspricht also einem für den Nutzer sichtbaren Element auf der abgebildeten Karte.

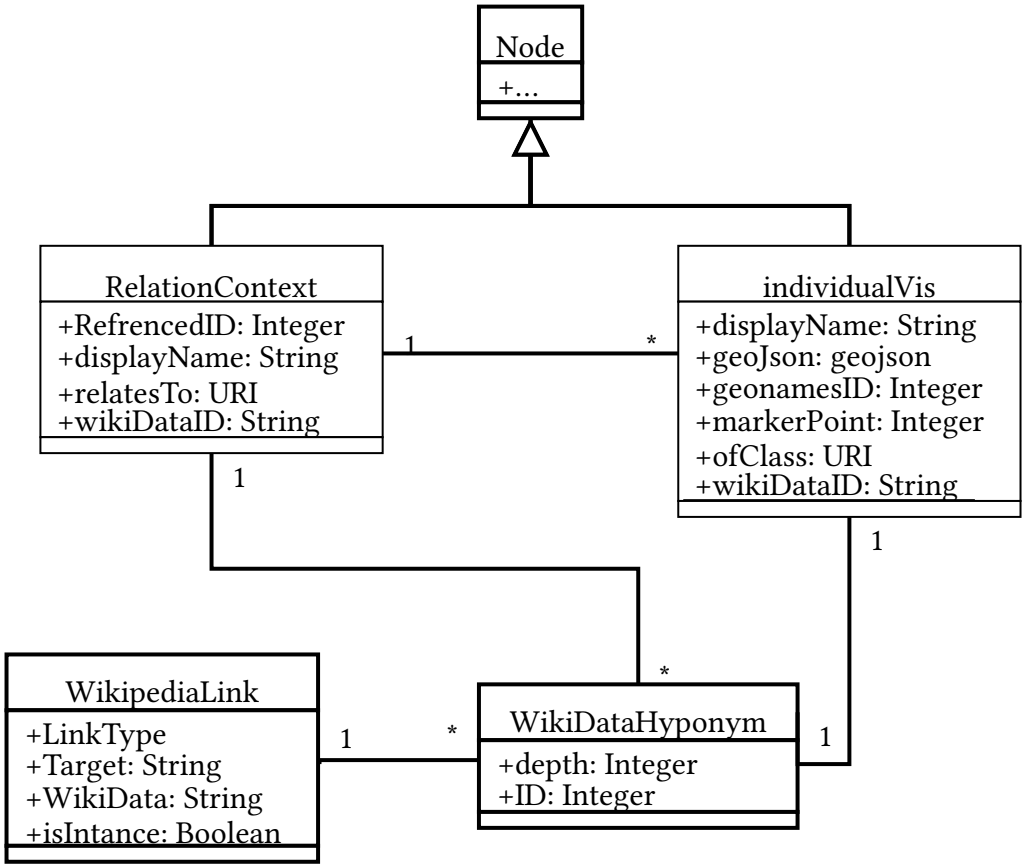


Abbildung 3.2.: Typesystem Struktur

4. FrontEnd

Das FrontEnd bildet die Schnittstelle zum Nutzer. In diesem Kapitel wird auf die Art der Visualisierung sowie ihre Nutzung eingegangen. Es soll außerdem ein Überblick über alle Funktionalitäten geboten und ihr Mehrwert erläutert werden.

Die Webapplikation zeigt den visualisierten Text in der Mitte und die Karte auf der rechten Seite an. So kann mit Text und Karte gleichermaßen gut interagiert werden. Die Karte basiert auf OpenStreetMap und bietet die Möglichkeit, zwischen der klassischen OpenStreetMap und einer Satellitenansicht zu wechseln. In dem in Abbildung 4.1 sichtbaren Panel befinden sich auf der rechten Seite alle visualisierten individuals. Um die Visualisierung zu ermöglichen, sind Daten notwendig, die in einem vorher durchgeführten Annotationsprozess aus dem Text gesammelt und mit weiteren Informationen angereichert wurden (siehe 5).
Im Folgenden wird auf die Features eingegangen, die von dem Endverbraucher in der Webapplikation selbst verwendet werden können.

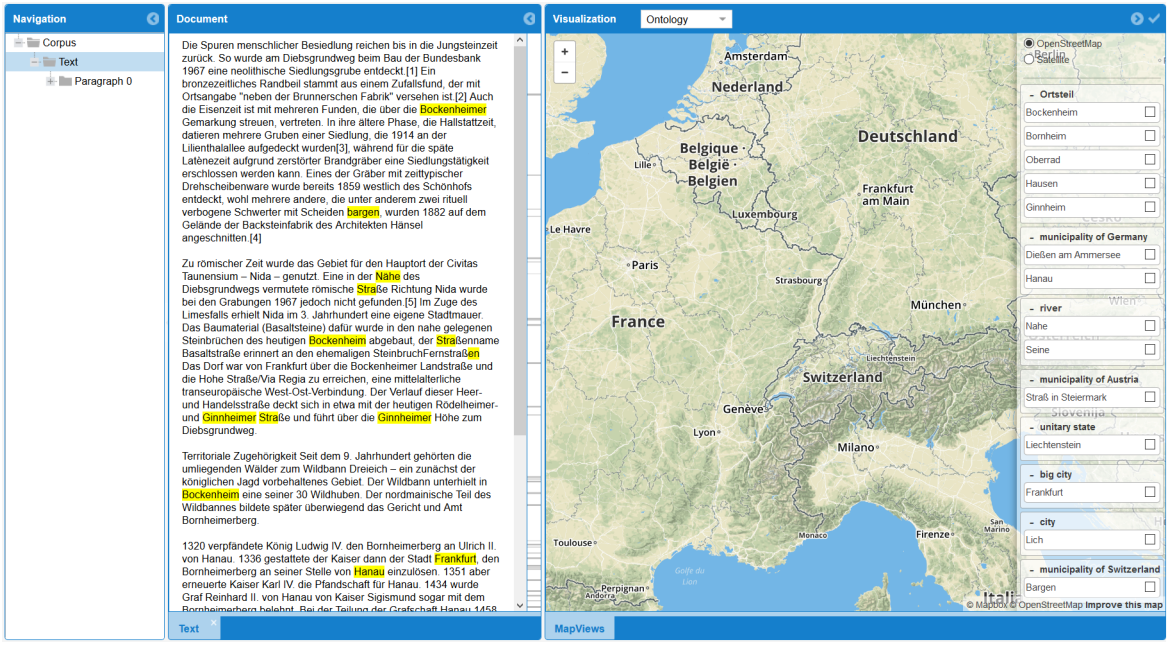
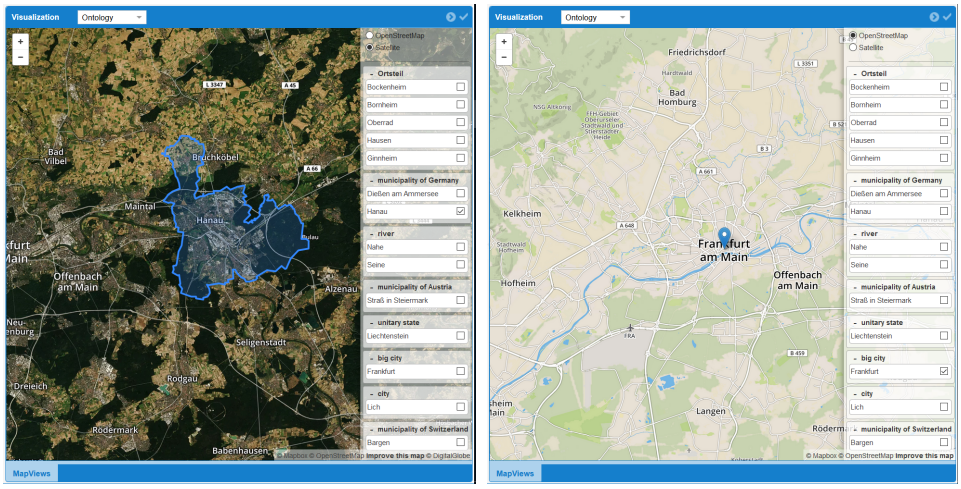


Abbildung 4.1.: GeoViz Tool

4.1. Visualisierung geographischer Informationen

Die Visualisierung geographischer Informationen verwendet Informationen aus den in Kapitel 5 genannten Vorverarbeitungsschritten, um für jedes individual entsprechende geographische Daten auf die Karte zu bringen. Diese werden wie in Abbildung 4.2b mit einem Marker oder wie in Abbildung 4.2a mit einer geographischen Fläche markiert dargestellt.



(a) Polygon Marker

(b) Point Marker

4.1.1. Popups

Mit einem Klick auf eines der Objekte öffnet sich wie in Abbildung 4.3 ein Popup mit Informationen zu dem entsprechenden Objekt. Neben einem Label und einem Bild sind auch eine WikiData ID mit einem Link zu der entsprechenden Seite sowie die Geonames ID und die URI der ontologischen Klasse, zu der das entsprechende individual gehört, enthalten. Die URI entspricht hierbei den im Panel dargestellten Gruppen und ist dementsprechend für jedes individual der gleichen Gruppe identisch.

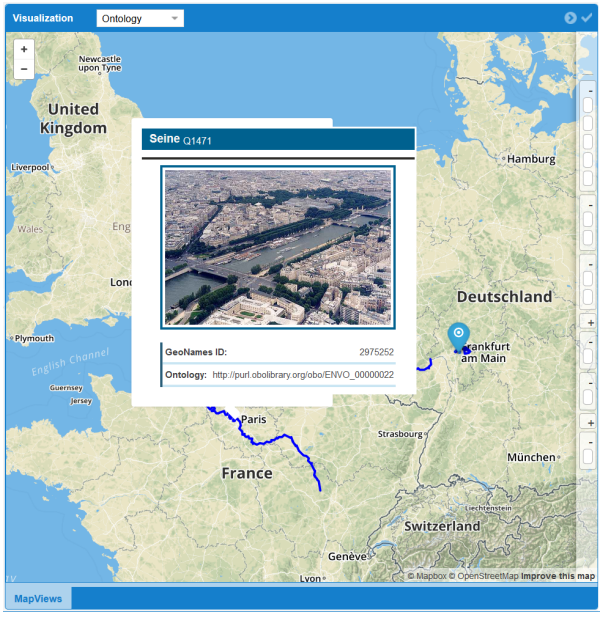


Abbildung 4.3.: Popup

4.1.2. Erkennung von Überschneidungen

Die Applikation bietet außerdem die Möglichkeit, Überschneidungen zwischen zwei Markierungen auf der Karte zu erkennen. Befindet sich eine der Markierungen ganz oder teilweise innerhalb einer zuvor ausgewählten Referenzmarkierung oder grenzt sie direkt an diese an, wird sie nicht blau, sondern grün hinterlegt (siehe Abbildung 4.4). Als Referenz wird immer die Markierung verwendet, die vom Nutzer als Erstes im Panel ausgewählt wurde. Das bietet die Möglichkeit, die im Text genannten Objekte auf ihren geographischen Zusammenhang zu prüfen. Näher eingegangen wird auf diese Funktionalität noch einmal in Kapitel 6.

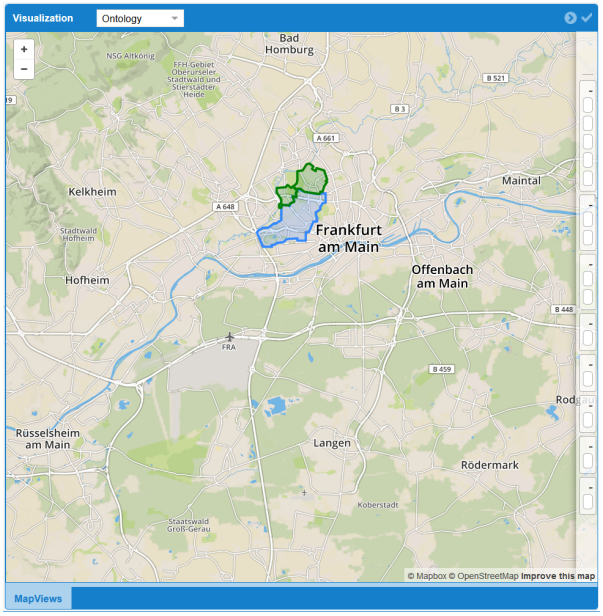


Abbildung 4.4.: Highlighting Map

4.2. Visualisierung des Textes

Der Text wird im mittleren Abschnitt des Webapplikation angezeigt. Hier sind alle visualisierten Token im Text gelb hinterlegt. So kann auf einen Blick gesehen werden, welche Objekte im Text visualisiert sind.

4.2.1. Scroll Panel

Neben dem Text findet sich ein Scroll Panel, in dem alle Stellen grau hervorgehoben sind, an denen die markierten Objekte liegen (siehe Anhang A.2). Wird eine Markierung auf der Karte angewählt, werden in dem Scroll Panel nur noch die Objekte angezeigt, die dieser Markierung entsprechen (siehe Anhang A.3). Die Markierung kann verwendet werden, um direkt zur gewünschten Stelle im Text zu springen. Mit einem Doppelklick kann der markierte Textteil grau hervorgehoben werden, um die Suche im Text zu vereinfachen.

4.2.2. Markierung von Wörtern

Für den Nutzer besteht auch die Möglichkeit, visualisierte Objekte im Text zu markieren (siehe Abbildung 4.5). Erkannt werden können diese daran, dass sie gelb hinterlegt sind. Wird ein Objekte markiert, so erhält diese Markierung einen Zeitstempel und eine ID. Alle vom User getätigten Markierungen stehen über einen Save Button zum Download zur Verfügung (siehe Abbildung A.1a u. A.1b). Das bietet dem Nutzer die Möglichkeit, den Verlauf all seiner im Text markierten Objekte anzusehen. Diese Funktion wird für die Evaluation in Kapitel 6 verwendet.

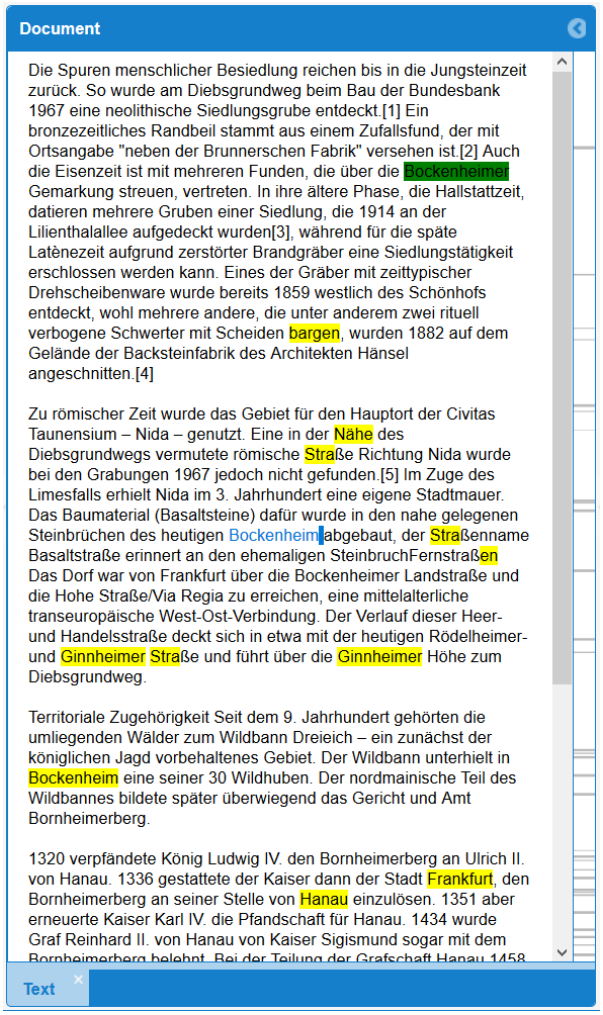


Abbildung 4.5.: HighlightingWord

5. BackEnd

Das BackEnd des Visualisationstools besteht aus zwei Hauptkomponenten. Die erste ist das ontologische Modell, auf dessen Basis die Strukturierung der Informationen stattfindet. Dieses muss für die geographische Visualisierung passend gewählt werden und wird vor Beginn der Annotation um WikiData IDs erweitert. Das gewährleistet die sprachunabhängige Identifikation der Klassen aus dem ontologischen Modell. Im zweiten Schritt erfolgt die Annotation der Informationen aus dem Text. Hierfür wird eine Annotator-Pipeline verwendet. Die für dieses Projekt geschriebenen Klassen bauen auf den in Abschnitt 2.2.1 beschriebenen Annotatoren auf.

5.1. Ontologisches Modell

In diesem Abschnitt wird darauf eingegangen wie das in Kapitel 3.2 beschriebene ontologische Modell in das vorliegende BackEnd geladen und für unser Programm optimiert wird.

5.1.1. OntologyStore

Zuerst werden über die PURL-Adresse die RDF-Informationen zu dem ontologischen Modell in die ontModel-Datenstruktur geladen. Es können mehrere Modelle in dieselbe Struktur eingefügt werden. Diese werden automatisch zusammengefasst. Anschließend wird zur Vorbereitung des Modells für jeden Eintrag eine WikiData-Anfrage (siehe Abschnitt 3.1) gesendet, um eine WikiData ID zu ermitteln und diese der ontologischen Klasse zuzuweisen. Über diese WikiData ID können anschließend sprachenunabhängige Suchen auf dem ontologischen Modell ausgeführt werden.

5.2. Annotatorpipeline

Die Annotator-Pipeline bildet das Grundgerüst eines Annotators. In ihr wird definiert, welche Annotationen in dem Programm ausgeführt werden sollen. Hierbei spielt die Reihenfolge der Annotatoren eine zentrale Rolle, da die erzeugten Annotationen oft als Input für weitere Annotatoren dienen. Sie bietet außerdem die Möglichkeit, einzelne Annotatoren auszutauschen oder nicht zu verwenden, solange die für die nachfolgenden Annotatoren benötigten Input-Daten vorhanden sind. Das bietet Entwicklern die Möglichkeit, andere Annotatoren zu verwenden oder eigene Annotatoren zu schreiben, um bessere Ergebnisse zu erzielen.
Die Annotator-Pipeline in dem vorliegenden Programm ist wie in Abbildung 5.1 aufgebaut. Informationen zu den von Drittanbietern erstellten Annotatoren finden sich in Kapitel 2. Im Folgenden wird auf die in blau hervorgehobenen Annotatoren eingegangen, da in diesen die vom FrontEnd verwendeten Datenstrukturen erzeugt werden.

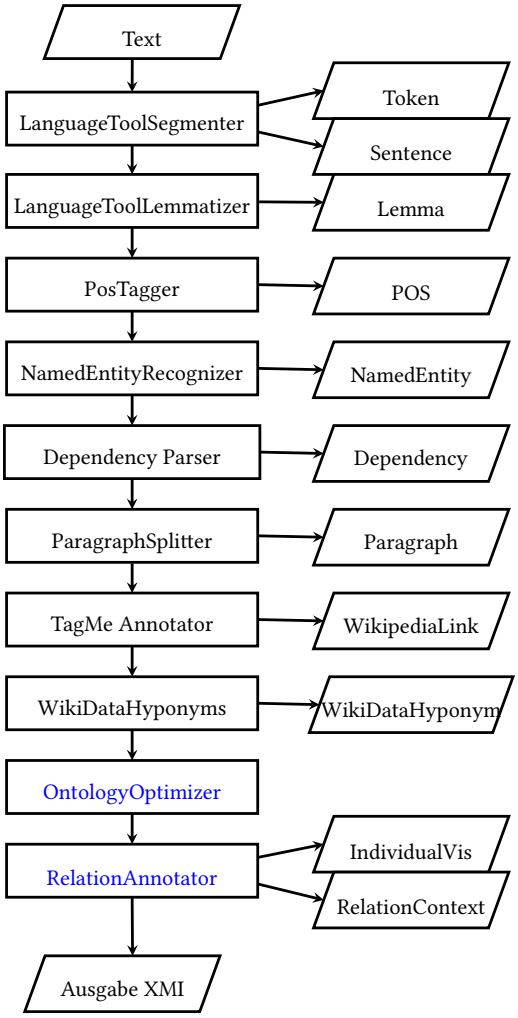


Abbildung 5.1.: Annotation Pipeline

5.2.1. OntologyOptimizer

Der OntologyOptimizer ist eine Klasse zur Erweiterung des ontologischen Modells auf Basis von aus dem Text gewonnenen Informationen. Für den Input verwendet die Klasse die Hyponyme aus dem WikiDataHyponyms Annotator (siehe Abschnitt 2.2.1). Der Annotator verwendet zwei Methoden, die während der Annotation aufgerufen werden.

classOptimizer

Mit der classOptimizer-Methode werden die Klassen des Modells um für den Text notwendige ontologische Klassen ergänzt. Es wird für jedes Hyponym per WikiData-Abfrage (s. Abschnitt 3.1) eine Überklasse ermittelt und geprüft, ob diese in dem verwendeten ontologischen Modell (s. Abschnitt 3.2) bereits existiert. Falls eine Überklasse besteht, geht das Programm weiter zum nächsten Hyponym. Existiert keine Klasse, wird die gefundene Klasse erzeugt und eine rekursive Funktion aufgerufen, die die geschilderten Schritte für die eingefügte Überklasse ausführt.

individualOptimizer

In der individualOptimizer-Methode wird per WikiData-Abfrage für jedes Hyponym ermittelt, ob es sich um ein individual handelt. Hierfür wird die WikiData Property ‚Instance of' verwendet. Besitzt ein WikiData-Objekt diese Property, handelt es sich um eine Named Entity, die einer Objektklasse zugeordnet werden kann. So wird eine Menge von Daten erhalten, die auch, aber nicht ausschließlich Objekte beinhaltet, die eine eindeutige geographische Zuordnung auf einer Karte aufweisen.

5.2.2. RelationAnnotator

Der RelationAnnotator ist eine Klasse zur Annotation der für die Visualisierung benötigten Informationen. Verwendet werden hierfür die in Abschnitt 2.1.1 erklärten Datentypen. Der Annotator hat drei Methoden, mit denen zwei verschiedene Annotationen generiert werden.

hyponymLinker

Die Hyponym-Linker-Methode erzeugt für jedes Hyponym einen Relation-Context-Eintrag. Dieser beinhaltet die WikiData ID und ein Label sowie einen Verweis auf die Überklasse des entsprechenden Hyponyms. Die Informationen erhält das Programm über eine SPARQL-Abfrage an das ontologische Modell.

classLinker

Die Class-Linker-Methode überprüft für die individuals aus dem ontologischen Modell, ob für ihre Klassen ein RelationContext exisitiert. Das ist notwendig, da dieser die Basis bildet, auf der in der Visualisierung die Klassen erstellt werden, denen die visualisierten individuals zugeordnet sind. Falls für eine der Klassen kein RelationContext existiert, wird dieser hier erzeugt. Die hierfür notwendigen Informationen werden dem ontologischen Modell entnommen.

individualVisualisier
Die IndividualVisualiser-Methode erzeugt für jedes individual, das in der Webapplikation visualisiert werden soll, ein individualVis-Objekt. In diesem Datentyp sind ein Label und die WikiData ID sowie die für die Visualisierung notwendigen geographischen Daten im geoJson-Format enthalten. Die geographischen Informationen werden über zwei Abfragen empfangen: eine SPARQL-Abfrage an WikiData und eine http.-Anfrage an Nominatim (s. Abschnitt 2.2.5). Die SPARQL-Abfrage findet für jedes Hyponym eine Geonames ID und eine Koordinate, die als Punkt für den Marker dienen kann. Über die Nominatim-Abfrage wird eine geoJson erzeugt, die verwendet wird, um eine geographische Markierung auf der Karte zu erzeugen (s. Abbildung 4.2a).

6. Evaluation

6.1. Aufgabenstellung

In diesem Kapitel wird die Evaluation des Visualisierungstools vorgenommen. Es handelt sich um eine Evaluation mit Testpersonen. Ihnen werden Texte vorgelegt, die auf zwei Arten bearbeitet werden sollten. Einer der Texte in dieser Arbeit soll mit dem TextToMap Annotator annotiert werden, der andere mit dem TextAnnotator (Abrami, Mehler u. a., 2019). (zunächst als TreeAnnotator (Helfrich u. a., 2018) veröffentlicht) Der Nutzer kann hierbei zwischen verschiedenen Annotatoren wählen und anschließend auf Basis eines vorher importierten UIMA-Typsystems eine manuelle, datenbankbasierte Annotation (Abrami und Mehler, 2018) durchführen. In diesem Fall wird dieses Framework verwendet, da keine mit dem Visualisierungstool vergleichbare Arbeit vorliegt und versucht wird, eine manuelle Annotation zu simulieren, die als Referenz dienen soll.

Die Aufgabe des Nutzers besteht darin, geographische Entitäten in den Texten zu finden und zu markieren. Hierfür werden dem Nutzer eine oder mehrere Eigenschaften genannt, die erfüllt werden müssen. Die Aufgabe im Beispieltext besteht darin, Frankfurter Ortsteile zu finden. In diesem Beispiel sind die zu beachtenden Eigenschaften, dass es sich um Frankfurter Ortsteile handelt die an Heddernheim angrenzen müssen. Der TextAnnotator dient hierbei als Referenzwert. Mit ihm soll die manuelle Suche nach den zu markierenden Entitäten simuliert werden. Anschließend werden die Tester gebeten, einen UMUX-Fragebogen auszufüllen, durch den in vier Fragen auf die Funktionalitäten und Benutzerfreundlichkeit des Visualisierungstools eingegangen wird.

6.2. Durchführung der Befragung

Das Ziel der Evaluation ist es, herauszufinden ob das Visualisierungstool dem Nutzer eine Visualisierung bietet, mit der er intuitiv die gegebenen geographischen Informationen auf den Text beziehen kann. Die Evaluation besteht für jeden Tester aus drei Teilen. Im ersten Schritt bekommt der Nutzer den beschriebenen Beispieltext und soll sowohl die Annotation mit dem TextToMap Annotator als auch mit dem TextAnnotator üben. Anschließend werden der Testperson zwei Texte vorgelegt, von denen jeder jeweils nur in einer der beiden Varianten annotiert ist. Hierbei erfolgt die Auswahl der Annotationsweise zufällig, sodass es für beide Texte und beide Varianten ausreichend Testergebnisse gibt.

Als Testpersonen werden Mitarbeiter und Praktikanten des TTLabs angefragt, die keinen Bezug zu der hier behandelten Arbeit haben.
Für die Evaluation wurden insgesamt 15 Testpersonen befragt.

6.3. Auswertung

Bei der Auswertung der Evaluation wird betrachtet, wie viele Fehler der jeweilige Proband gemacht hat und wie lange er gebraucht hat. Als Fehler gelten sowohl Annotationen, die nicht der gestellten Aufgabenstellung entsprechen, als auch fehlende Annotationen. Die Dauer bezieht sich auf die Dauer, die der Nutzer benötigt hat, um alle Entitäten im Text zu markieren. Hierbei wird nicht beachtet, ob der Nutzer die Entitäten vorher schon kannte oder diese erst noch ermitteln musste. Für die Annotation mit dem Visualisierungstool durfte der Nutzer ausschließlich die im Tool vorhandenen Funktionalitäten nutzen. Für die Annotation in TextAnnotator durften die Testpersonen alle ihnen zugänglichen Quellen nutzen. Das beinhaltet sowohl beliebige Internet-Suchmaschinen als auch Hilfsmittel wie Stift und Papier.

Der UMUX-Fragebogen beinhaltet Fragen, die auf einer Skala von eins bis sieben beantwortet werden sollen. Hierbei bedeutet eins ‚Strongly Disagree' und sieben ‚Strongly Agree'.

6.4. Ergebnisse

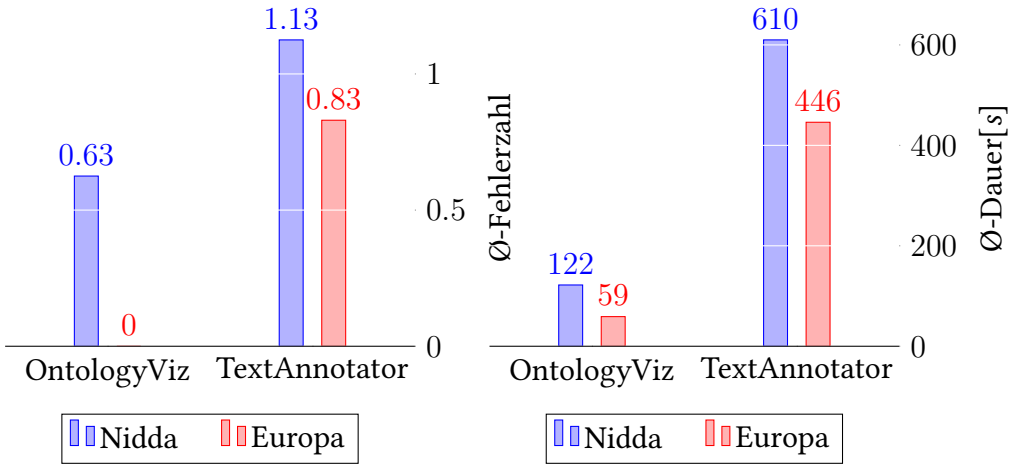
Für die Evaluation wurde 15 Testpersonen die beschriebene Aufgabenstellung vorgelegt. Diese wurde von allen vollständig erfüllt. Die in diesem Abschnitt erläuterten Ergebnisse stehen in Beziehung zueinander und stammen aus den genannten 15 Evaluationsdurchläufen.

6.4.1. Annotation

In dem ersten hier dargelegten Beispiel geht es um einen Text, in dem alle Ortsteile annotiert werden sollten, die an Nidda grenzen. In diesem Text liegt der Fokus eher auf der Quantität der Annotationen im Text als auf der Schwierigkeit. Im Vergleich zum zweiten Text lässt sich erkennen, dass fast alle Testpersonen in der Lage waren, die gefragten Ortsteile zu identifizieren. Bei Verwendung von TextToMap Annotator wurden im Schnitt 0,63 Fehler pro Annotationsdurchlauf gemacht, bei der manuellen Annotation mit TextAnnotator 1,13 Fehler (siehe Abbildung 6.1a). Die manuelle Annotation konnte in diesem Beispiel also ähnlich gute Ergebnisse erzielen. Ein größerer Unterschied ist bei der in Abbildung 6.1b dargestellten Zeit erkennbar. Nutzer des Textes brauchten im Schnitt 122 Sekunden, Nutzer des TextAnnotators im Schnitt 610 Sekunden. Da Nutzer des Visualisierungstools einen direkten Bezug zwischen allen geographisch visualisierten Entitäten und den entsprechenden Stellen im Text erkannten, konnte Zeit gespart werden. Die Fehlerzahl von 0,63 bei Verwendung des TextToMap Annotators lässt aber Spielraum zur Verbesserung, da viele Tester davon ausgegangen sind, dass jeder Ortsteil nur einmal im Text vorkommen würde und die vom Tool bereitgestellte Markierung der Textstellen nicht eindeutig genug ist. Auffällig war, dass bei der Benutzung des zweiten Tools verschiedene Herangehensweisen angewandt wurden. Die Testpersonen haben sich entweder an der im Panel festgelegten Reihenfolge oder der geographisch gegeben Reihenfolge entlang gearbeitet. Die Dauer war bei beiden Herangehensweisen ähnlich.

Im zweiten Beispiel wurde den Testpersonen ein Text über die Geographie Europas gegeben. Ihre Aufgabe war es, jeden Gletscher zu markieren, der sowohl in Italien als auch in Frankreich liegt. Es waren nur zwei Gletscher im Text enthalten, die diesen Anforderungen

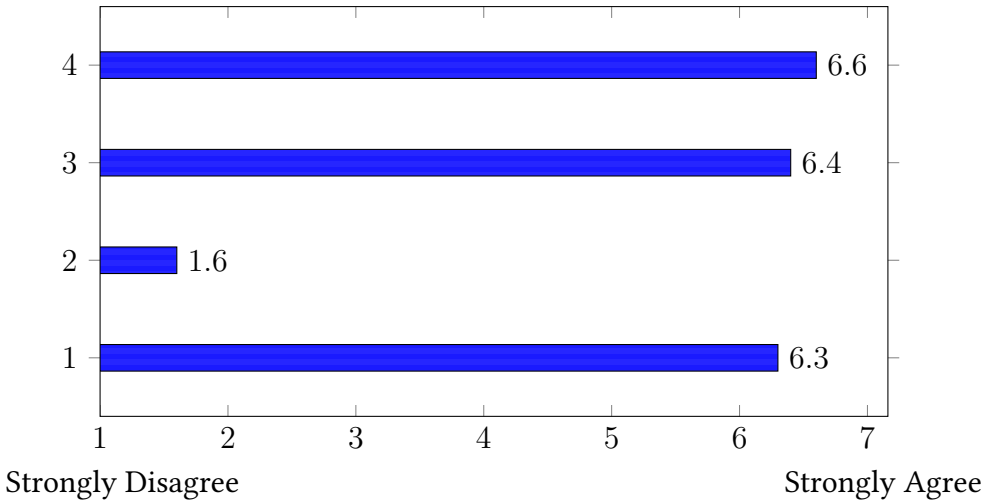
entsprechen. Der Fokus lag hier auf der Aufgabe, diese Gletscher zu identifizieren und anschließend im Text zu markieren. An dem in Abbildung 6.1a dargestellten Diagramm kann erkannt werden, dass in TextToMap Annotator alle Gletscher korrekt markiert wurden, während die durchschnittliche Fehlerzahl bei der manuellen Annotation bei 0,83 liegt. Da für die Gletscher oft nur das Land, zu dem sie offiziell gehören, aufgeführt wurde, ist es bei manuellen Annotation öfter zu Fehlern gekommen. Bei der Verwendung des Tools konnten die Testpersonen mit Hilfe der eingezeichneten Position der Gletscher und den Landesgrenzen genau bestimmen, ob Teile eines Gletschers in beiden Ländern liegen. Bei der Zeitdauer ist ein ähnliches Ergebnis zustande gekommen wie bei dem ersten Text. Da die gefragten Gletscher im Text nah beieinander liegen und keiner der beiden doppelt im Text vorgekommen ist, hat die Annotation in TextToMap Annotator im Schnitt nur 59 Sekunden gedauert. In TextAnnotator lag die durchschnittliche Dauer bei 446 Sekunden, da die Testpersonen durch die fehlende Vertrautheit mit dem Thema eine vorsichtigere Herangehensweise gewählt haben.



(a) Diagramm: Durchschnittliche Fehler (b) Diagramm: Durchschnittliche Dauer

6.4.2. UMUX-Fragebogen

Der UMUX-Fragebogen dient dazu, einen Eindruck darin zu bekommen, wie die Tester die Anwendung und ihre Funktionalitäten einschätzen. Hier werden nach dem UMUX-Standard vier Fragen gestellt. Aus dem in Abbildung 6.2 dargestellten Diagramm lässt sich herauslesen, dass die Tester das Tool als nützlich und einfach handhabbar empfinden.



4. The data is visualized in a clear and understandable way.
3. The visualisation tool is easy to use.
2. Using the visualisation tool is a frustrating experience.
1. The visualisation tool's functionalities meet my requirements.

Abbildung 6.2.: Auswertung: UMUX Fragebogen

6.4.3. Schlussfolgerung

Das Tool bietet die nötigen Funktionalitäten, um geographische Informationen auf für den Nutzer verständliche Art zu visualisieren. Es bietet die Möglichkeit, durch direktes visuelles Feedback geographische Zusammenhänge verständlich zu machen, und setzt sie in Bezug zum Text. Durch die oben genannten Vorteile lässt sich vor allem eine starke Beschleunigung im Vergleich zur manuellen Annotation erkennen. Die Fehlerzahl beider Annotationen variiert mit der Menge der zu annotierenden Wörter und der Vertrautheit der Nutzer mit dem Themenbereich. Das macht die Applikation vor allem für Nutzer sinnvoll, die keine Kenntnisse in den geographischen Bereichen haben, die im Text behandelt werden.

Die Evaluation zeigt außerdem, dass der TextToMap Annotator verwendet werden kann, um Annotationen auf einem Text zuverlässig durchzuführen. Auf sich dadurch eröffnende Anwendungsgebiete wird in Abschnitt 7.4 eingegangen. Empfunden wird das Tool als nützlich

und übersichtlich, vor allem bei Texten, die Vorwissen oder einen hohen Suchaufwand erfordern.

7. Fazit und Ausblick

In folgendem Kapitel findet sich im Fazit eine Zusammenfassung der Arbeit in ihren einzelnen Schritten und die erzielten Ergebnisse. Anschließend wird auf mögliche Anwendungsgebiete und mögliche zukünftige Projekte eingegangen.

7.1. Fazit

Das Ziel der vorliegenden Arbeit war es, ein Programm zu entwickeln, das auf Basis ontologischer Modelle geographische Informationen aus einem Text auf einer Karte visualisiert. Diese Visualisierung soll Nutzern ohne Voraussetzung von Kenntnissen im Bereich Informatik die Möglichkeit geben, automatisch geographische Informationen aus Texten einzulesen und in verständlicher Weise darzustellen. Hierfür wurde ein in Javascript entwickeltes Front-End verwendet, das auf einen Blick den Text und eine OpenStreet-Map-Karte darstellt. Die visualisierten Entitäten sind sowohl im Text als auch auf der Karte hervorgehoben, um den Nutzern eine Verbindung zwischen Text und Bild zu bieten. So werden Informationen wie die geographische Lage und auch Zusammenhänge zwischen den geographischen Entitäten für den Nutzer verständlich dargestellt.

Die hierfür benötigten Informationen stammen aus einem für dieses Projekt entwickelten Annotator. Um die Kompatibilität mit anderen in Text Technology Lab entwickelten Frameworks zu gewährleisten, arbeitet er auf Basis eines UIMA-Typsystems. Zu Anfang der Arbeit wurden verschiedene ontologische Modelle dahingehend untersucht, ob sie ausreichend viele Klassen besitzen, in denen geographische Informationen repräsentiert werden. Von den untersuchten ontologischen Modellen wurden zwei ausgewählt und in einer lokalen TDB-Datenbank gespeichert, um als Grundlage für die Klassifizierung der annotierten Tags zu dienen.

Im zweiten Schritt wurde eine Annotator-Pipeline aufgebaut. Die Voraussetzung war hier, dass ein Datentyp existiert, der die Worte aus dem Text repräsentiert, die mit den Klassen aus dem ontologischen Modell abgeglichen werden können. Als Basis hierfür dient die TagMeAnnotator API. Das ist ein von Drittanbietern entwickelter Annotator, der über eine API-Abfrage Worte in einem Text erkennt und mit ihren jeweiligen Wikipedia-Artikeln verlinkt. Erweitert wurde dieser durch einen in TTLab entwickelten Annotator, den WikiDataHyponyms Annotator. Dieser verwendet die Ausgabe der API, um entsprechende WikiData-Einträge zu ermitteln und sie in dem WikiData-Hyponyms-Datentyp zu speichern. Um eine Verbindung zwischen diesen und dem ontologischen Modell zu schaffen, wurde jede Klasse des ontologischen Modells um ihre entsprechende WikiData ID erweitert.

Auf Basis der im Text ermittelten Informationen und der WikiData-Einträge wurde das ontologische Modell optimiert, um alle für die Visualisierung nötigen Klassen zu enthalten. Abschließend wurden im letzten Annotationsschritt auf Basis des optimierten ontologischen Modells und den WikiData Hyponyms alle Daten gespeichert, die für die Visualisierung benötigt werden.

Zum Ende der Arbeit wurde eine Evaluation ausgeführt. In dieser sollten sowohl Ergebnisse zur Funktionalität als auch zur Nutzerzufriedenheit ermittelt werden. Am deutlichsten wird hierbei die Zeitersparnis, die das Tool dem Nutzer bringen kann. Da die Entitäten aus dem Text eine geographische Visualisierung erhalten, kann der Nutzer die Lage sowie Proximitäten und weitere Zusammenhänge zu anderen Objekten direkt aus der Applikation lesen. Im Test konnten die gestellte Aufgabe mit Verwendung des TextToMap Annotators im Schnitt in nur 17 % der Zeit im Vergleich zur manuellen Annotation ausgeführt werden. Bei der Genauigkeit war er vor allem dann im Vorteil, wenn es sich um Themenbereiche handelte, die dem Nutzer nicht vertraut waren. In einer UMUX-Umfrage haben alle Testpersonen angegeben, das Tool einer manuellen Annotation vorzuziehen.

7.2. Anwendungsmöglichkeiten

Der TextToMap Annotator wurde als eigenständige, modulbasiertes Software entwickelt. Das bedeutet die Anwendungsmöglichkeiten beziehen sich nicht nur auf das Programm selbst sondern auch auf die einzelnen Module die getrennt voneinander funktionieren können. Die im Kapitel 5 beschriebenen Annotatoren können beispielsweise als weiteres Glied in eine Pipeline inkludiert werden, solange die benötigten Annotationsschritte darin enthalten sind. Auch das Backend benötigt als Eingabe nur eine XMI in der die passenden Datentypen enthalten sind. Diese können bspw. auch von Hand annotiert sein.

7.2.1. TextImager

Das entwickelte Tool kann als eine Erweiterung für den TextImager verwendet werden. In dem Text To Map Annotator wird mit der Visualisierung geographischer Informationen ein Bereich abgedeckt, der im TextImager noch nicht enthalten ist. Es wird dem Text Imager also eine zusätzliche Funktion geboten, die helfen kann, Nutzern Informationen bereitzustellen.

7.2.2. TextAnnotator

Da der TextAnnotator auf dem UIMA-Typsystem aufbaut, könnte der TextToMap Annotator auch hier verwendet werden, um zukünftig das Erkennen und Annotieren von geographischen Informationen zu vereinfachen. Durch eine geographische Repräsentation der Informationen muss während der Annotation keine weitere Recherche betrieben werden. Es wird außerdem sichergestellt, dass die Informationen aktuellen Standards entsprechen. Der Nutzer könnte also Zeit sparen und sichergehen, dass alle Annotationen den definierten Standards entsprechen.

7.3. Desiderata (Offene Probleme)

7.3.1. Ontologisches Modell

Das ontologische Modell bildet den Kern dieses Projektes. Da ontologische Modelle oft unter Berücksichtigung eines bestimmen Zwecks entwickelt werden sind diese nicht für die Anwendung in diesem Projekt optimiert. Es fehlen viele individuals und Klasseen, die für die Visualisierung nötig sind. Diese werden in der Applikation per WikiData optimiert. Da WikiData eine Wissensdatenbank ist, die auf eine Struktur wert legt die Zusammenhänge hervorheben will, kommen so oft Klasseen Zustande die für eine Visualisierung nicht optimal sind. Ein Ansatz zur Lösung des Problems wäre, die Applikation mit ausreichend Texten durchlaufen zu lassen und anschließend das erzeugte ontologische Modell per Hand zu optimieren. Da das Modell gespeichert wird wäre das eine einmalige Aufgabe.

7.3.2. TagMeAnnotator

Der Großteil der fehlerhaften Annotationen im Text entsteht durch schwer einzuordnende Wörter, wie bspw. Frankfurt oder seine. Der TagMeAnnotator erkennt in manchen Fällen nicht die korrekte Named Entity oder ordnet sogar Pronomen als Named Entity (z. B. seine = der Fluss Seine) ein. Da der Text in einzelnen Paragraphen gesendet wird, kann hier noch mit der Länge gearbeitet werden, um die Ausgabe zu optimieren. Außerdem arbeitet das A^3 Lab, in dem der TagMe Annotator entwickelt wurde, an weiteren Entity Linker Annotatoren. In der Zukunft könnte der TagMeAnnotator also auch von einem Nachfolger abgelöst werden. Als Beispiel dient das WAT-Projekt, das zur Zeit der Erstellung dieser Arbeit nur englische Texte verarbeiten kann.

7.4. Future Work

In diesem Abschnitt werden mögliche Erweiterungen und Anwendungsmöglichkeiten für das in dieser Arbeit behandelte Projekt und seine Komponenten vorgestellt.

7.4.1. Optimierung des geocodings

Das vorliegende Projekt arbeitet ausschließlich mit einer Nominatim-Suche. Diese wird durch die WikiData ID unterstützt, um zu garantieren, dass die korrekte geographische Entität visualisiert wird.

Die Suche lässt sich aber auf zwei Arten optimieren:

Da nicht für jedes Objekt in der Nominatim-Abfrage ein Repräsentation der Fläche als geoJson verfügbar ist, muss in vielen Fällen auf Marker zurückgegriffen werden. Durch das Hinzufügen weiterer Geocoding-Dienste könnte eine bessere Rate an visualisierbaren Flächen erreicht werden.

Die zweite Verbesserungsmöglichkeit besteht in der Menge der Daten, die bei der Abfrage verwendet werden. Wie oben erwähnt wird zurzeit eine WikiData ID verwendet, um die Suche zu optimieren. Durch die Erweiterung der Suche mit Informationen wie dem Land, in dem das individual liegt, aber auch der Klasse, zu der es gehört, können die Laufzeit und die Genauigkeit erhöht werden.
Beispiele für weitere Suchmaschinen für das Geocoding sind:

1. OpenCage
2. Photon
3. GeoCheck
4. WhatisWhere

Es müsste gegebenenfalls vorher überprüft werden, welche dieser Suchmaschinen Open-Source-Projekte sind.

7.4.2. Bereitstellung zusätzlicher Operationen

In der Visualisierung besteht die Möglichkeit, dem Nutzer mehr Operationen zur Verfügung zu stellen. Hierzu könnte das Markieren kompletter Klassen gehören oder eine Visualisierung geographischer Entitäten, die Koordinaten mit einer ausgewählten Entität teilen.

7.4.3. Klassifikation von Texten

Die in diesem Projekt verwendeten Annotatoren machen keinen Unterschied zwischen geographischen und nicht geographischen Objekten. Die Filterung dieser findet erst statt, sobald versucht wird, für das jeweilige Objekt eine entsprechende geoJson zu finden. Es besteht die Möglichkeit, auf Basis des zugrunde liegenden ontologischen Modells einen Text einzulesen und alle gefundenen Tags zu klassifizieren. Das würde die Anzahl der Nomninatim-Abfragen reduzieren und zu einer Verbesserung der Laufzeit führen

7.4.4. Ontology matching

Denkbar ist es auch, die von den Annotatoren erzeugten Klassen zu verwenden, um zwei ontologische Modelle auf Basis eines Textes abzugleichen. So könnte gezielt für die im Text angesprochenen Themen die Struktur innerhalb verschiedener ontologischer Modelle erkannt werden.

7.4.5. Erweiterung der visualisierten Struktur

In der vorliegenden Version des Programms werden individuals in ihre entsprechenden ontologischen Klassen eingeordnet. Diese Gruppen werden anschließend visualisiert. Mit den extrahierten Daten besteht die Möglichkeit, das Panel und damit die für den Nutzer sichtbare Struktur zu erweitern, um Zusammenhänge zwischen visualisierten Objekten noch eindeutiger darzustellen.

7.4.6. Erweiterung der Querverweise

Innerhalb des ontologischen Modells können Querverweise integriert werden, die grundlegende Informationen, wie bspw. das Land oder die Regierung, unter dessen/deren Jurisdiktion eine geographische Entität liegt, einfach abrufbar machen.

7.4.7. Visualisierung historischer Texte

Der zugrunde liegende Annotationsprozess kann auch Informationen zu historischen Entitäten sammeln und speichern. Allerdings können bei ihrer Visualisierung Diskrepanzen auftreten, da die verwendete Karte der Repräsentation einer aktuellen Weltkarte entspricht. Sowohl natürliche als auch anthropogene territoriale Entitäten können sich über die Zeit verändern. Die Funktion, diese Informationen aus dem Text zu lesen und in der Visualisierung zu repräsentieren, könnte das Tool auch für die Auswertung historischer Texte anwendbar machen. Zu beachten ist hier die Abfrage der geoJson-Dateien, da diese der verwendeten Karte entsprechen müssen.

Glossar

Annotator Annotatoren sind Werkzeuge die im Natural Language Processing verwendet werden um spezifische Merkmale in Texten verarbeiten, und als Metadaten an das Dokument anhängen. 8, 16, 25, 26, 28, 33, 37

CAS Das Common Analysis System nach (Gotz und Suhre, 2004) ist eine Untersystem des UIMA Frameworks, das den Datenaustausch zwischen den verschiedenen Komponenten im Framework regelt. Es bietet Funktionen für Typisierungen und die Annotation von Texten und Datenzugriffen. 10, 16, 37

individual Als individual wird in diesem Projekt eine Entität in einem ontologischen Modell bezeichnet, das eine Named Entity repräsentiert und einer ontologischen Klasse zugeordnet ist. 11, 20, 21, 26, 27, 35–38

Klasse Als eine class wird im gegebenen Fall eine Entität in einem ontologischen Modell bezeichnet, die eine Gruppe von Individuen beinhalten kann. 11, 33–37

Lemma Ein Lemma beschreibt den Wortstamm eines Wortes. 13, 37

Named Entity Der Begriff Named Entity bezeichnet Wörter, die durch Eigennamen identifiziert werden können. In dem vorliegenden Projekt werden geographische Entitäten (bspw. Mount Everest) gemeint. 12, 14, 26, 35, 37

OWL Die W3C Web Ontology Language (*OWL - Semantic Web Standards* 2019), kurz OWL, ist eine zur Repräsentation komplexer Daten wie Gruppen und Relationen entwickeltes 'semantic web language'. 12, 16, 37

Panel Mit einem Panel ist hier eine Schaltfläche gemeint, die dem Nutzer die Möglichkeit gibt, verschiedene visualisierte Objekte auf der Karte ein- und auszublenden. Die Informationen, die im Panel angezeigt werden, stammen aus einem Vorverarbeitungsschritt im BackEnd-Teil dieses Projekts und entsprechen den individuals. 22, 36, 37

PURL Ein PURL wird im gegebenen Text als Abkürzung für die Internationalized Resource Identifier der ontologischen Modelle verwendet. Über die PURL-Adressen wird das ontologische Modell abgerufen und in den lokalen Speicher geladen. 16, 24, 37

RDF Das Resource Description Framework (*RDF - Semantic Web Standards* 2019) ist ein Modell zum Austausch von Daten im Web. Das RDF bietet einen Standard zum verbinden von Daten, der unabhängig von der, den Daten zu Grunde liegenden, Struktur verwendet werden kann. 16, 17, 24, 37

Token Der Begriff Token beschreibt eine Einheit, die durch die in der Annotation verwendeten Regeln für die Tokenisierung definiert wird. In dem vorliegenden Projekt entsprechen Tokens einzelnen Wörtern und Satzzeichen. 13, 14, 22, 37

UMUX Die Usability Metrics for User Experience (UMUX) umfassen nach (Finstad, 2010) eine Methode zur Auswertung der von einem Tester empfundenen Nutzbarkeit einer Anwendung. 28, 29, 31, 34, 37

Literatur

Abrami, Giuseppe und Alexander Mehler (2018). „A UIMA Database Interface for Managing NLP-related Text Annotations“. In: *Proceedings of the 11th edition of the Language Resources and Evaluation Conference, May 7 - 12*. LREC 2018. Miyazaki, Japan.

Abrami, Giuseppe, Alexander Mehler, Andy Lücking, Elias Rieb und Philipp Helfrich (2019). „TextAnnotator: A flexible framework for semantic annotations“. In: *Proceedings of the Fifteenth Joint ACL - ISO Workshop on Interoperable Semantic Annotation, (ISA-15)*. ISA-15. Gothenburg, Sweden.

Buttigieg, Pier Luigi, Norman Morrison, Barry Smith, Christopher J Mungall und Suzanna E Lewis (2013). „The environment ontology: contextualising biological and biomedical entities“. In: *Journal of biomedical semantics* 4.1, S. 43.

Clemens, Konstantin (2015). „Geocoding with openstreetmap data“. In: *GEOProcessing 2015*, S. 10.

Fekete, Jean-Daniel, Jarke J. van Wijk, John T. Stasko und Chris North (2008). „The Value of Information Visualization“. In: *Information Visualization: Human-Centered Issues and Perspectives*. Hrsg. von Andreas Kerren, John T. Stasko, Jean-Daniel Fekete und Chris North. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–18. ISBN: 978-3-540-70956-5. DOI: 10.1007/978-3-540-70956-5_1. URL: https://doi.org/10.1007/978-3-540-70956-5_1.

Ferragina, P. und U. Scaiella (2012). „Fast and Accurate Annotation of Short Texts with Wikipedia Pages“. In: *IEEE Software* 29.1, S. 70–75. DOI: 10.1109/MS.2011.122.

Finstad, Kraig (2010). „The Usability Metric for User Experience“. In: *Interacting with Computers* 22.5, S. 323–327. ISSN: 0953-5438. DOI: 10.1016/j.intcom.2010.04.004. eprint: http://oup.prod.sis.lan/iwc/article-pdf/22/5/323/1992916/iwc22-0323.pdf. URL: https://doi.org/10.1016/j.intcom.2010.04.004.

Gotz, T. und O. Suhre (2004). „Design and implementation of the UIMA Common Analysis System“. In: *IBM Systems Journal* 43.3, S. 476–489. DOI: 10.1147/sj.433.0476.

Helfrich, Philipp, Elias Rieb, Giuseppe Abrami, Andy Lücking und Alexander Mehler (2018). „TreeAnnotator: Versatile Visual Annotation of Hierarchical Text Relations“. In: *Proceedings of the 11th edition of the Language Resources and Evaluation Conference, May 7 - 12*. LREC 2018. Miyazaki, Japan.

Hemati, Wahed, Tolga Uslu und Alexander Mehler (2016). „TextImager: a Distributed UIMA-based System for NLP“. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*. Osaka, Japan: The COLING 2016 Organizing Committee, S. 59–63. URL: https://www.aclweb.org/anthology/C16-2013 (besucht am 13. 09. 2019).

Liddy, E.D. (2001). „Natural Language Processing“. In: *Encyclopedia of Library and Information Science* , 2nd Ed.

Manning, Christopher u. a. (2014). „The Stanford CoreNLP natural language processing toolkit“. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, S. 55–60.

OWL - Semantic Web Standards (2019). URL: https://www.w3.org/OWL (besucht am 23. 08. 2019).

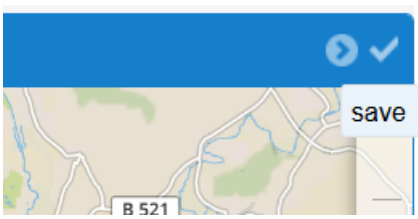
RDF - Semantic Web Standards (2019). URL: https://www.w3.org/RDF (besucht am 23. 08. 2019).

The DKPro Core Team (2019). URL: https://dkpro.github.io/dkpro-core/releases/1.9.3/docs/component-reference.html (besucht am 13. 09. 2019).

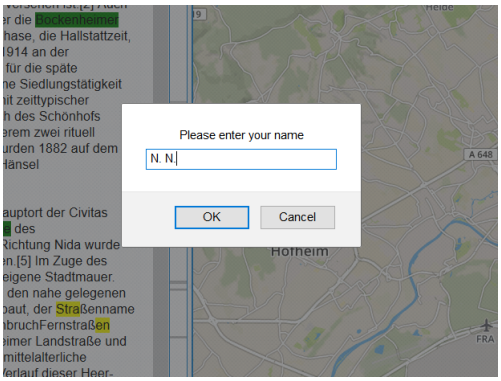
Vrandečić, Denny und Markus Krötzsch (2014). „Wikidata: A Free Collaborative Knowledge Base“. In: *Communications of the ACM* 57, S. 78–85. URL: http://cacm.acm.org/magazines/2014/10/178785-wikidata/fulltext (besucht am 13. 09. 2019).

Weiland, Claus u. a. (2018). „BioFID, a platform to enhance accessibility of biodiversity data“. In: *Proceedings of the 10th International Conference on Ecological Informatics*. Jena, Germany. URL: https://www.researchgate.net/profile/Marco_Schmidt3/publication/327940813_BIOfid_a_Platform_to_Enhance_Accessibility_of_Biodiversity_Data/links/5bae3e3e92851ca9ed2cd60f/BIOfid-a-Platform-to-Enhance-Accessibility-of-Biodiversity-Data.pdf?origin=publication_detail.

A. Abbildungen



(a) Save Button



(b) Name prompt

Abbildung A.1.: SaveButoon & NamePrompt

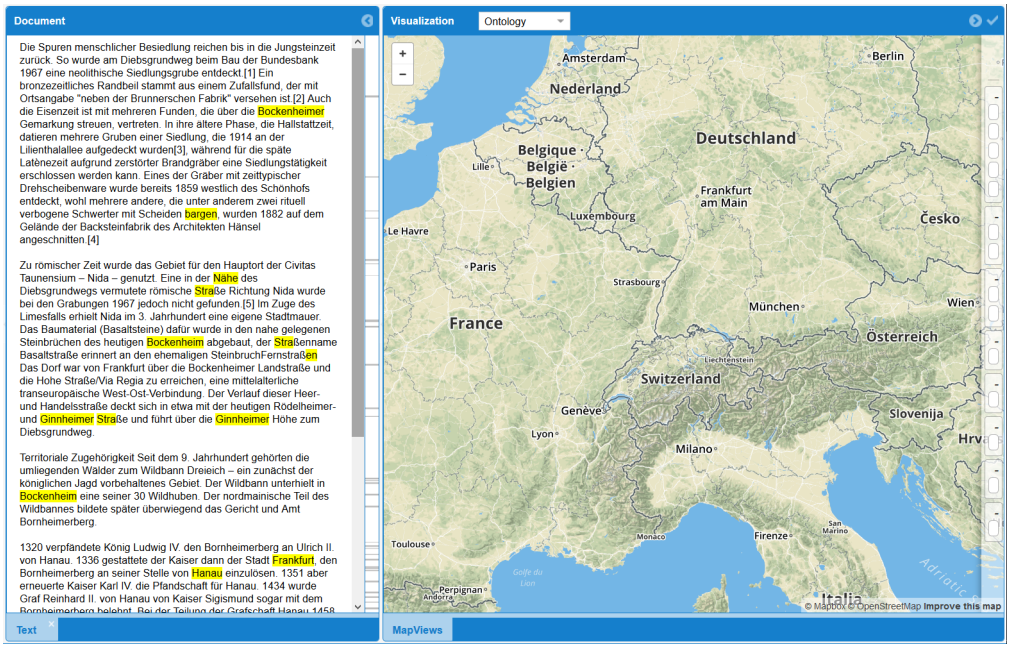


Abbildung A.2.: Scroll Panel

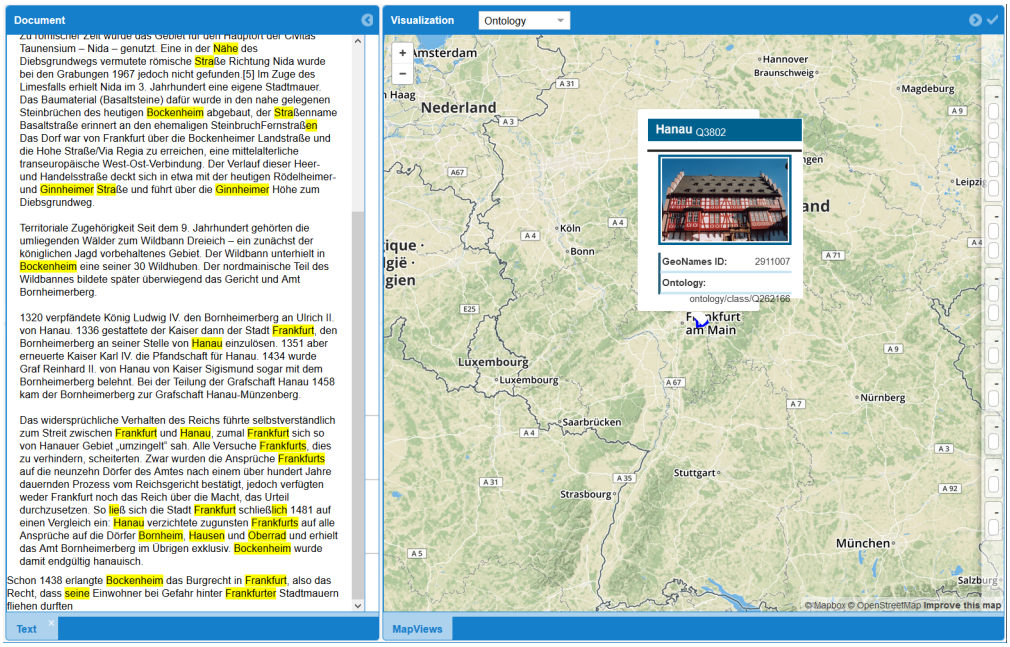


Abbildung A.3.: Scroll Panel for 1 object

B. README

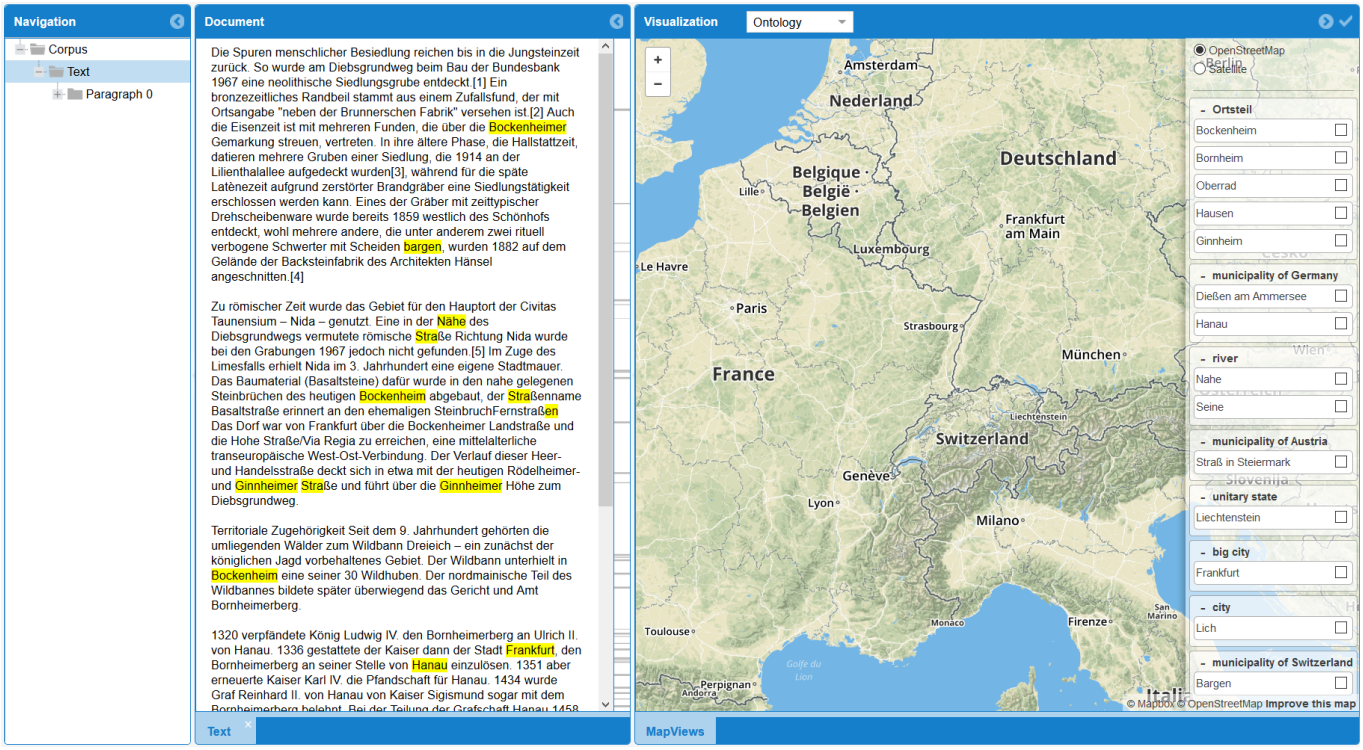
Die folgenden zwei Seiten zeigen die README Datei des Projektes, wie es auf der beigefügten CD enthalten ist.

TextToMap Annotator

The TextToMap Annotator is application used for an automated viusalisation of a text corpus by marking geographical features.

Built based on MapBox it supports the visualisation of areas and Points.

The ontology model used to structure the information can be changed depending on the context



Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

The only requirement for this project is a local TDB storage

To install with Node.js:

npm install tdb

Installing

1. Open the OntologyStore class and add the ontology models you want to use.

```
 OntModel ontoModel =
ModelFactory.createOntologyModel(OntModelSpec.RDFS_MEM_RDFS_INF, null);
```

```
        ontoModel.read("http://purl.obolibrary.org/obo/geo/nation-
geography.owl");
        ontoModel.read("http://purl.obolibrary.org/obo/geo/basic-
geography.owl");
        ontoModel.read("http://purl.obolibrary.org/obo/envo.owl");
```

2. There are a number of different ways to run the Annotator

   The easiest way to run the annotator is to set the cas variable in the main class to the text you want to annotate.

   ```
    JCas cas = JCasFactory.createText("Johann Wolfgang Goethe ist 1749 in
   Frankfurt am Main geboren.", "de");
   ```

   After this is done you can run the annotator to generate a js document that will be visualized

   It is possible to run the annotator as a webservice by running the webservice class. It will be available under the local host with port: 4567.

3. The visualisation can be set up using the index.html

   The visualisation will update automatically after every new annotation.

Built With

- Maven - Dependency Management
- TDB - RDF Storage

Authors

- **Pedro Alves Zipf**

Acknowledgments

- Special thanks to Prof. Dr. Alexander Mehler and Giuseppe Abrami
- Hat tip to anyone whose code was used