HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Tag-Suchmaschine und Thesaurus für OpenStreetMap

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2014

Autor: Simon Gwerder
Betreuer: Prof. Stefan Keller

Impressum

Autor
Simon Gwerder
Sentmattstrasse 45
CH-8912 Obfelden
sgwerder@hsr.ch

Institut
Hochschule für Technik Rapperswil
Oberseestrasse 10
CH-8640 Rapperswil
office@hsr.ch

Revision

Datum	Dokumentteil	Version	Änderung
22.09.2014	Teil I Teil II Teil III	1.0 1.0 1.0	Dokument erstellt, vorläufige Kapitelstruktur
03.10.2014	Teil I Teil III	1.1 1.1	Bearbeitung Einführung Bearbeitung Projektorganisation, -management
04.10.2014	Teil II	1.1	Bearbeitung Anforderungsspezifikation: Use Case Diagramme
05.10.2014	Teil II	1.2	Bearbeitung Anforderungsspezifikation: Aktoren, Use Cases
06.10.2014	Teil II	1.3	Bearbeitung Anforderungsspezifikation: Use Cases
01.11.2014	Teil III	1.4	Zeitplanung
20.11.2014	Teil I	1.5	Kapitel Attributive Heterogenität
25.11.2014	Teil IV	1.5	Verschiedenste Anhänge
09.12.2014 - 21.12.2014	Teil I Teil II Teil III	2.0	Restliche Kapitel, Dokumentation abgeschlossen

Tabelle: Dokumentversionen

Abstract

Die Studienarbeit „Tag-Suchmaschine und Thesaurus für OpenStreetMap“ beschäftigt sich mit den frei erstellbaren Tags von OpenStreetMap. Diese Tags dienen zur Markierung interessanter Objekte in der Karte. Aktuell gestaltet sich die Suche nach den Tags umständlich. Einerseits sind die Suchfunktionen sehr limitiert, andererseits wird die Suche durch die attributive Heterogenität erschwert. So führen zum Beispiel Mehrdeutige Konzepte, zeitliche Entwicklung und geographische Unterschiede zur Schaffung von Tags, welche synonym sind oder Ober- und Unterbegriffe beschreiben.

Die Webapplikation „TagFinder“ bietet hier eine intuitive Suchmaschine an. Anders als die bestehenden Lösungen ermöglicht „TagFinder“ die Suche nach Begriffen, die den Tag beschreiben. Der Tagname oder Teile davon müssen nicht bekannt sein. Als Basis dient ein Thesaurus im Semantic Web Format (RDF/XML), der zusätzliche Begriffsinformationen enthält und diese untereinander in Beziehung bringt. Mit Hilfe einer selbst entwickelten Kommandozeilen-Applikation kann dieser Thesaurus unterstützend aufgebaut werden. Zudem können die Daten durch die Integration des Webdienstes TagInfo automatisch um Tag-Statistiken und neue Tags aktualisiert werden.

Das Backend des „TagFinder“ Systems wurde in Python und dem Microframework Flask realisiert und ermöglicht ein Hosting auf allen WSGI-fähigen Plattformen wie z.B. Heroku. Desweiteren erlauben WebServices die Verwendung aller Funktionen der Suchmaschine in eigenen Applikationen. Im Frontend wurde auf ein einfaches, klares Design mit Bootstrap gesetzt.

Die Webanwendung ist unter http://tagfinder.herokuapp.com erreichbar.

Management Summary

Ausgangslage

Damit in OpenStreetMap interessante Orte wie Kinos, Restaurants und Wanderwege richtig dargestellt werden können, werden die Stellen auf der Karte von freiwilligen Mappern mit sogenannten Tags versehen. Die Tags geben den Objekten erst eine Bedeutung.
Jedoch is die Suche nach Tags in OpenStreetMap aktuell sehr umständlich. Den Erfassern muss der Tagname oder Teile davon bekannt sein. Ist das nicht der Fall, müssen sie eine grosse Liste von bekannten Tags durchforsten. Hinzu kommt die Problematik der attributiven Heterogenität.
Ziel dieser Studienarbeit ist es eine Webseite zu realisieren, die eine dedizierte Such-maschine für OpenStreetMap Tags anbietet. Diese Suchmaschine soll speziell Neulingen bei OpenStreetMap das Auffinden von Tags erleichtern.

Vorgehen

Für die Planung des Projekts TagFinder wurde das Vorgehensmodell RUP (Rational Unified Prozess) gewählt, welche eine inkrementelle, iterative Softwareentwicklung erlaubt.
Zunächst musste die zu Verfügung stehende Projektzeit in Phasen aufgeteilt und eine Grobplanung für die Arbeitspackete erstellt werden. Bei der Planung musste eine längere Einarbeitungszeit berücksichtigt werden, da erst die Programmiersprache Python erlernt und grundlegendes Wissen um Begriffsnetzwerke angeeignet werden musste.
Anschliessend ging es daran, so schnell als möglich einen einfachen Thesaurus zu erstellen, da er die Basis der Suchmaschine darstellt. Mit Hilfe der Webservices von TagInfo konnte das Vorhaben zügig umgesetzt werden und ein erster Prototyp war vorzeigefähig.
In der nächsten Phase konnte der Thesaurus um die wichtigen Begriffsbeziehungen erweitert werden. Leider war es nur möglich dies manuell zu bewerkstelligen, da im Bereich der Begriffsnetzwerke auf wenig zurückgegriffen werden konnte. So musste jeder Key und Tag im Basis-Thesaurus einzeln betrachtet und Synonyme, Ober- und Unterbegriffe in Deutsch und Englisch hinzugefügt werden. Zur erleichterten Erfassung wurde ein einfaches Kommandozeilen-Tool entwickelt, dass mit Hilfe von Online-Wörterbüchern wie OpenThesaurus oder Wordnik Vorschläge zu den Begriffen heraussucht. Dennoch, die Bearbeitung des Thesaurus war sehr aufwenig und nahm viel Zeit in Anspruch: Es wurde schnell klar, dass bis Ende des Projekts nicht alle Tags durchgearbeitet werden konnte und die Priorität musste auf den Information Retrieval Prozess, den Suchalgorithmus und die restlichen Anforderungen, wie Webservices und Website-Design gesetzt werden.

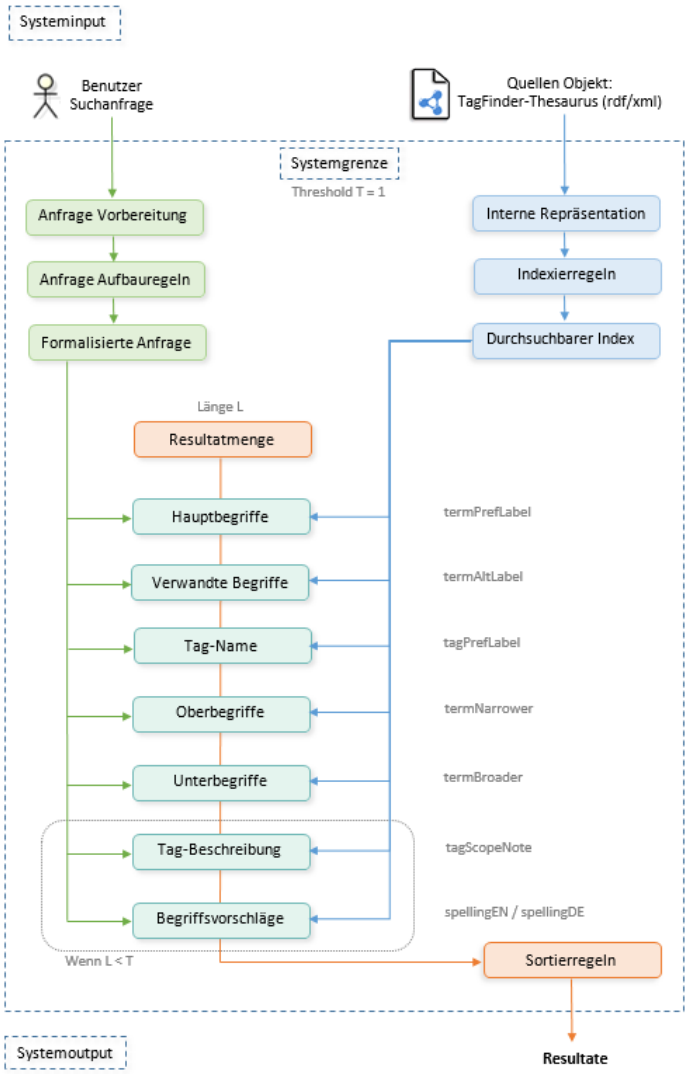


Abbildung: Der Suchprozess von TagFinder

Ergebnisse

Als Ergebniss der Arbeit kann die TagFinder Webseite mit einer voll funktionsfähigen Suchmaschine abgeliefert werden. Die Suchmaschine basiert auf einem umfangreichen Thesaurus, welcher mit eigenen Tools aktualisiert und weiter verbessert werden kann.
Es konnte am Schluss die gesetzten Ziele erreicht werden. Vorallem die TagFinder Webseite ist sehr gelungen und es konnten sogar einige optionale Features implementiert werden. Nicht ganz vollständig muss jedoch der TagFinder Thesaurus abgegeben werden. Aufgrund des aufwendigen Erstell- und Bearbeitungsprozesses konnten nur die wichtigsten Keys und Tags um verwandte Begriffe, Ober- und Unterbegriffe erweitert werden. Jedoch sind auch die unbearbeiteten Tags besser auffindbar als bei den existierenden Tag-Suchen. Bei Tests konnte gezeigt werden, das die Suchmaschine fähig ist, für nahzu alle Begriffe eines Editor Presets die passenden Tags zu finden.

Ausblick

Im Rahmen dieses Projekts konnte ich mir viele neue Fähigkeiten aneignen. Dazu gehört neben Python auch der Umgang mit modernen Webtechnologien. Die Beschäftigung mit den Begriffsnetzwerken war sehr spannend, haben diese doch viel mit künstlicher Intelligenz zu tun.
Die TagFinder Webseite und die Suchmaschine stehen gut da. So habe ich bereits positive Feedbacks aus der OpenStreetMap Community erhalten und er wurde sogar im deutschsprachigen OSM Wochenblog erwähnt. Dennoch sehe ich einigen Raum für Weiterentwicklungen. Ein nützliches Feature wäre das Hinzufügen einer Sortierfunktion, die erlaubt alle Resultate nach Trefferscore oder komplett nach Statistiken der Tags zu sortieren. Weitere Verbesserungen wären im Bereich des Suchprozesses und der Vorschläge möglich: So hat der TagFinder z.B. noch Schwierigkeiten im Umgang mit Suchbegriffen im Plural.

Beim Thesaurus sollten die restlichen Tags und Keys nachgeführt werden. Ist dieser Schritt getan, lässt sich mit dem Netzwerk auch komplexe Analysen der Tags und Begriffen anstellen. Auch eine Homogenisierung der OSM-Daten wäre dann denkbar.

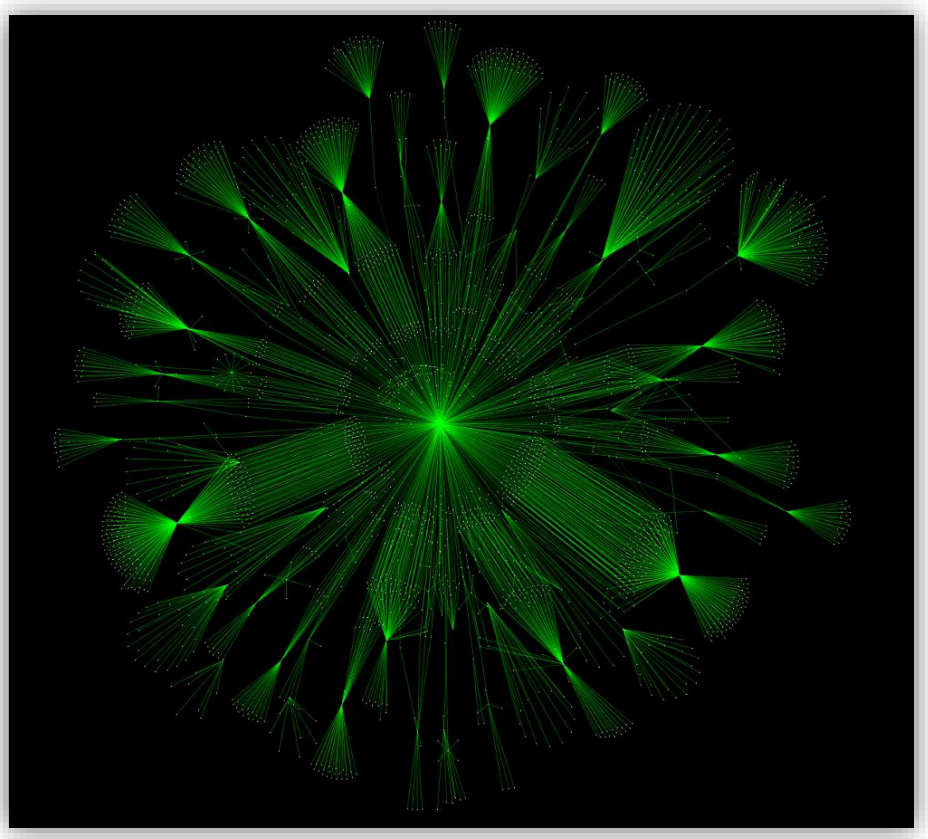


Abbildung: Übersicht über den TagFinder Thesaurus Graph. Punkte stellen OSM-Tags / -Keys oder Begriffe dazu dar. Verbunden werden sie mit hierarchischen Relationen.
Im Mittelpunkt steht das Konzept „RelatedTerm“. (Der Graph wurde mit Cytoscape 3.2.0 berechnet)

Urheber- und Nutzungsrechte

Vereinbarung

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit „Tag-Suchmaschine und Thesaurus für OpenStreetMap“ von Simon Gwerder unter der Betreuung von Prof. Stefan Keller geregelt.

2. Urheberrecht

Die Urheberrechte stehen dem Student zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von dem Student wie von der HSR nach Abschluss der Arbeit verwendet und weiter entwickelt werden

Rapperswil, den..... .....

Der Student

Rapperswil, den..... .....

Der Betreuer der Studienarbeit

Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, den..... .....

Der Student

Inhaltsverzeichnis

Teil I: Technischer Bericht

Teil II: SW-Projektdokumentation

Teil III: Projektmanagement

Teil IV: Anhang

Teil I: Technischer Bericht

1 Einführung

OpenStreetMap ist ein Projekt, das zum Ziel hat gemeinsam eine umfangreiche Karte der ganzen Welt zu erstellen. Anders als zum Beispiel Google Maps zählt OpenStreetMap vollständig auf interessierte Freiwillige, welche die Karte kontinuierlich erweitern und verbessern. Sie können neue Strassen, Gebäude, Fuss- und Wanderwege, Parks und vieles mehr erfassen. Es ist sogar möglich eigene Attribute, sogenannte Tags zu erstellen um neue Kartenobjekte zu benennen. Dies führt in OpenStreetMap zu einem grossen Detailreichtum, so gibt es in OpenStreetMap mehrere tausend verschiedene Objektarten. Das ist ein Vorteil gegenüber Behördendaten und kommerziell erfassten Daten.

Diese Freiheiten haben aber auch Nachteile:
Objekte können unterschiedlich attribuiert werden, obwohl sie dasselbe bedeuten. Zu bestehenden Tags können also Synonyme erstellt werden. Desweiteren gibt es für die Tags keine hierarchische Struktur mit Oberbegriffen, um eine Gruppe zusammenfassen zu können.

Aus Sicht des Kartenerfassers wird es bei der riesigen Auswahl an Tags immer schwieriger herauszufinden, wie ein reales Objekt richtig attribuiert werden soll. Ausserdem gibt es aktuell nur eine exakte Suche für die Liste der bekannten Tags.
Andererseits ist es auch problematisch für die Nutzer alle Kartenobjekte eines Typs zu finden, wie zum Beispiel alle Kleidershops inklusive Modegeschäfte, Boutiquen und Schuhläden. Um die Daten zu professionellen Zwecken nutzen und analysieren zu können, braucht es eine definierte Qualität und eine Begriffshierarchie.

Ziel dieser Studienarbeit ist es eine Webseite zu realisieren, die eine dedizierte Such-maschine für OpenStreetMap Tags anbietet. Diese Suchmaschine soll speziell Neulingen bei OpenStreetMap das Auffinden von Tags erleichtern. Erfahreneren Mappern und Entwicklern ermöglicht das neue Begriffsnetzwerk eine weitgehende Analyse und Homogenisierung von OSM-Daten.

Abb. I-1: Das Logo von TagFinder

1.1 Aufgabenstellung und Rahmenbedingungen

Diese Arbeit wird im Rahmen einer Studienarbeit (SA) an der Hochschule für Technik Rapperswil (HSR) durchgeführt. Die Aufgabenstellung ist vorgegeben und kann im Kapitel IV-7 im Detail eingesehen werden.
Als Grundlage für die Suchmaschine wird ein umfassender Thesaurus aufgebaut, welcher die Tags in eine Begriffshierarchie fügt und somit ein eigentliches Tag-Netzwerk darstellt. Als Datenquelle für den Thesaurus können auch OSM-Daten-Statistiken und Wörterbücher genutzt werden.

Lieferobjekte:

- Tag-Suchmaschine (Website)
- Thesaurus (z.B. im RDF/XML-Format) mit Werkzeugen zur Aktualisierung
- Webservice zur Tag-Suchmaschine (RESTful)
- Anhand exemplarischer Beispiele soll - als Ergebnis der Nachbereitung des Thesaurus - die attributive Heterogenität aufgezeigt werden
- Die vom Studiengang geforderte Dokumentation

Rahmenbedingungen:

- Webtechnologien: JavaScript, HTML5, RESTful Webservice
- Programmiersprache: Python, ev. Java
- Der Student entscheidet sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer vereinbart.
- User Interface, Source Code, Code-Kommentare Versionsverwaltung und Installations-anleitungen sind in Englisch. Alles andere ist deutsch.

1.2 Arbeitstitel

Im Folgenden wird der Arbeitstitel „TagFinder“ verwendet:

Titel	Bezeichnung
TagFinder	Bezeichnung für das Gesamtsystem.
TagFinder - Website	Bezeichnet die Webanwendung der Suchmaschine. (Verfügbar auf: http://tagfinder.herokuapp.com)
TagFinder - Webservice	Steht für die Webservices der Suchmaschine. (Verfügbar auf: http://tagfinder.herokuapp.com/api)
TagFinder - Thesaurus	Steht für den Thesaurus und dessen Updating Werkzeuge

Tabelle I-1: Arbeitstitel des Projekts

1.3 Aufbau der Arbeit

Die Dokumentation dieser Arbeit ist in vier Teile gegliedert:

Teil	Beschreibung
Teil I: Technischer Bericht	Der erste Teil der Dokumentation gibt eine Einführung in das Projekt, zeigt die Ziele und Bedingungen, den Stand der Technik, das Vorgehen und Resultate der Arbeit
Teil II: SW-Projektdokumentation	Dieser Teil beinhaltet Software Engineering spezifische Dokumentationen. Dazu gehören die Analyse mit Anforderungsspezifikation, Domain, Packages und Klassen, Website-Design, der Suchprozess und Testing.
Teil III: Projektmanagement	Hier werden die Planung und das Monitoring des Projekts aufgeführt.
Teil IV: Anhang	Der Anhang umfasst: Inhalt der CD, das Glossar, Literatur- und Quellenverzeichnis, eine Installationsanleitung (in Englisch), das Webservice API, die Aufgabenstellung und die Reviews.

Tabelle I-2: Aufbau der Arbeit

2 Technisches Umfeld und Begriffe

2.1 OpenStreetMap

OpenStreetMap (OSM) ist ein wiki-artiges Projekt, um gemeinsam eine Karte der gesamten Welt zu erstellen. Es ist die bekannteste Alternative zu Google Maps, die mit ihren weltweit über 1.9 Millionen Nutzern (Stand 2014) ein riesiges Potential hat [1]. Die Zahl der Nutzer wird weiter steigen (Siehe Abb I-2).

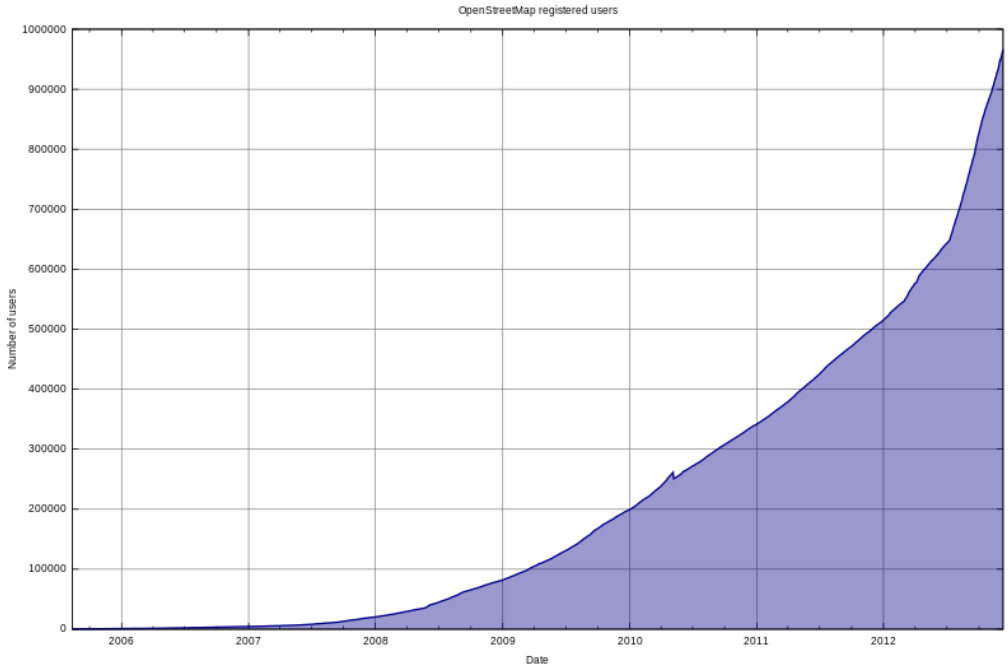


Abb. I-2: Anzahl registrierte OSM Benutzer in den Jahren 2006-2013 [2]

Da die freiwilligen Mapper ihr lokales Wissen einbringen, hat OSM – im Unterschied zu Behördendaten und kommerziell erfassten Daten – einen viel höheren Detailreichtum mit über tausend Objektarten. Ein weiterer Vorteil ist die offene Lizenz [3] der Kartendaten, was zu unzähligen spezialisierten Anwendungen führt: Apps für Wanderrouten, Webseiten für die Wohnungssuche, Restaurant Guides, Navigationssoftware, etc.
Der Aufbau der Karten ist einfach, es gibt drei Grunddatenelemente: Punkte (Nodes), Linien (Ways) und Relationen (Relationens). Aus diesen Elementen sind alle Objekte aufgebaut. Eine Fläche (Area) lässt sich zum Beispiel mit einer geschlossenen Linie realisieren.

Um alle möglichen Formen bilden zu können stellt das OpenStreetMap-Projekt eine Vielzahl an Bearbeitungswerkzeugen zur Verfügung. Am weitesten verbreitet sind die Editoren JOSM, iD und Potlatch 2 [4]. Bei JOSM besteht sogar die Möglichkeit eigene Plugins zu entwerfen.

2.2 Tags

Um den Datenelementen überhaupt eine Bedeutung zu geben, können sie mit ein oder mehrere sogenannte Tags versehen werden. Auf Deutsch werden Tags auch als Attribute bezeichnet. Diese bestehen aus einem Key (Schlüssel) und einer Value (Wert), wobei Key und Value durch ein Gleichheitszeichen getrennt werden.

Abb. I-3: Tag-Symbol

- Der Key beschreibt in den meisten Fällen eine Oberklasse der beschriebenen Eigenschaft.
- Der Value spezifiziert die Eigenschaft genauer.

Eine ungeschriebene Regel lautet, dass nur englische Keys und Values erstellt und verwendet werden sollen. Oft hält sich die Community an diese Regeln und es gibt fast keine Ausnahmen.
Ein Beispiel für einen Tag ist `amenity=cinema`, welcher ein Kino bezeichnet. Möchte man ein Punkt oder eine Fläche auf der Karte als Spielplatz markieren kann man `leisure=playground` benutzen.
Je nach Applikation, welche die Karte rendert, werden die Tags in Piktogramme umgewandelt, sodass Nutzern schnell „Point of Interests“ (POI‘s) erkennen können:

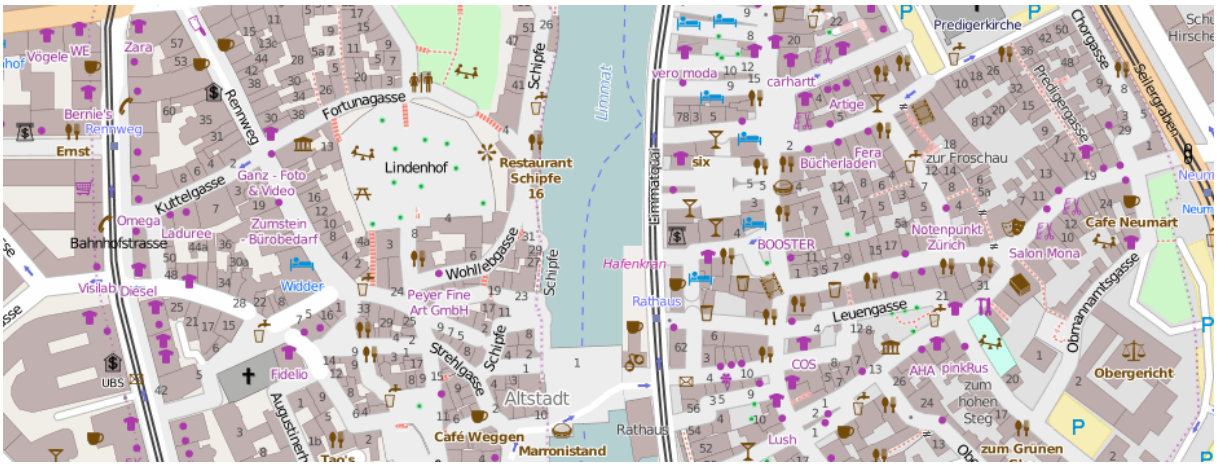


Abb. I-4: Kartenauschnitt aus OSM mit vielen POI’s. Man sieht drei Spielplätze (leisure=playground).

Nun gibt es eine Vielzahl an Tags, mit denen man nur eine bestimmte Eigenschaft bezeichnen kann. Fügt man diese Tags Objekten hinzu, die bereits mit einem „Haupttag“ markiert sind, werden sie zu dessen „Untertags“. Der Haupttag beschreibt das Objekt als Ganzes oder allgemeiner und die Untertags nur spezielle Aspekte davon. Man spricht in diesem Fall auch von einem „Tagging-Schema“. Ein gutes Beispiel sind Gebäude: Als Haupttag wird `building=yes` verwendet. Als Untertags kann zum Beispiel mit `addr:street=*` die Strasse angegeben werden, an der das Gebäude steht, mit `addr:housenumber=*` die Hausnummer und mit `building:levels=*` die Anzahl Etagen des Gebäudes. (Sterne sind Platzhalter für Values)

2.3 Thesaurus

Ein Thesaurus ist eine Sammlung an Begriffen aus einer Sprache. Die Begriffe können über Beziehungen miteinander verbunden werden, es bildet sich ein Wortnetz. Es hat sich gezeigt, dass Thesauri ein geignetes Hilfsmittel für die Recherche darstellen. Ein Thesaurus bildet ein „kontrolliertes Vokabular“, bei dem jeder Begriff eindeutig benennt werden kann. Begriffe mit leicht unterschiedlicher Schreibweise (z.B. Foto / Photo), Synonyme, Abkürzungen oder Übersetzungen, etc. haben alle ihre eigene Identifikation und sind über assoziative oder hierarchische Relationen vernüpft. Dies ermöglicht das Indexieren, Speichern und Finden von Dokumenten. Bei einer Recherche nach einem Begriff, können über die Relationen alle passenden Benennungen gefunden werden. Es gibt Äquivalenz-, Assoziations- und hierarchische Relationen [5]. Die wichtigsten Relationen in Thesauri sind:

Relation	Relationstyp	Erläuterung
Synonym	Äquivalenzrelation: reflexiv, symmetrisch, transitiv	Beziehung zwischen Begriffen mit gleicher oder sehr ähnlicher Bedeutung (verwandte Begriffe): z.B. Vieh <--> Tier
Oberbegriff	Hierarchische Relation: irreflexiv, asymmetrisch, transitiv	Beziehung von allgemeineren, umfassenderen Begriffen zu Unterbegriffen: z.B. Tier --> Säugetier
Unterbegriff	Hierarchische Relation: irreflexiv, asymmetrisch, transitiv	Beziehung von spezialisierteren, engeren Begriffen zu Oberbegriffen: z.B. Katze --> Säugetier

Tabelle I-3: Die wichtigsten Relationen in Thesauri

Schnell wurde die Nützlichkeit von Thesauri in der Informatik entdeckt, um menschliches Wissen repräsentieren zu können. So ist zum Beispiel das Wort „Venus“ mit unterschiedlichen Semantiken belegt. Für den Menschen kann „Venus“ einerseits eine römische Göttin bedeuten, andererseits ein Planet in unserem Sonnensystem. Eine Suchmaschine hingegen hätte ohne Thesaurus, in dem er das Wort nachschlagen könnte, keine Möglichkeit diese Zuordnungen herzustellen. In der Informatik spricht man in diesem Zusammenhang auch von Semantic Web [6] oder Ontologie. Jedoch sollte man den mehrfach belegten Ausdruck „Ontologie“ vermeiden. [7]

Die Beziehungsnetze können auch als Graph dargestellt werden. Dabei bilden Begriffe die Knoten und Relationen verbinden diese als Kante. Ist die Relation asymmetrisch, ist die Kante gerichtet.

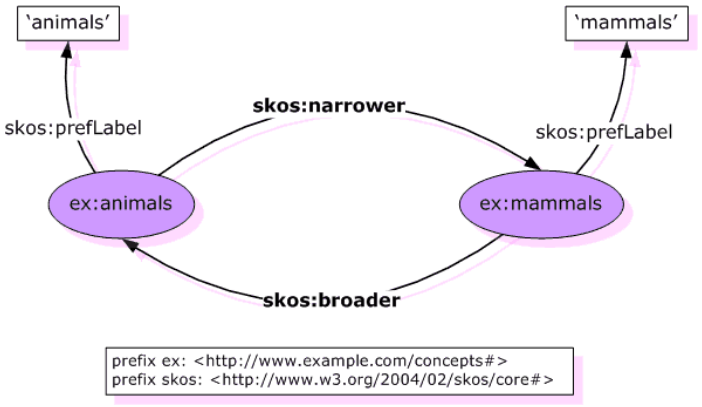


Abb. I-5: Tier - Säugetier Beziehung im SKOS Schema als Graph dargestellt.

Vereinheitlicht werden die Graphen mit XML Schemas. So definiert das Schema SKOS einen Namensraum für Thesaurus Relationen (Siehe Abb I-5) [8]. Zum Beispiel würde für den Begriff. „Tier“ ein eigenständiges SKOS-Konzept angelegt. Mit `skos:prefLabel` wird dem Konzept die Benennung „Tier“ gegeben. Allfällige Synonyme wie z.B. „Vieh“, können dann per `skos:altLabel` angehängt werden. Eine Benennung durch `skos:prefLabel` wird gegenüber solchen mit `skos:altLabel` bevorzugt.
Es besteht die Möglichkeit ein Thesaurus Graph mit dem Resource Description Framework (RDF) standardisiert zu serialisieren: Die Daten werden in sogenannten „Triples“ abgespeichert. Ein Triple besteht aus einem Subjekt, einem Prädikat und einem Objekt. Für SKOS wird dabei ein ein Triple pro asymmetrischer Relation und zwei pro symmetrischer Relation serialisiert.

Subjekt	Prädikat	Objekt
ex:animals	skos:prefLabel	„animals“
ex:animals	skos:broader	ex:mammals
ex:mammals	skos:prefLabel	„mammals“
ex:mammals	skos:narrower	ex:animals

Tabelle I-4: Umwandlung der Beziehungen in Abbildung 4 zu Tripplen.

Heutzutage wurden für die verschiedensten Anwendungsgebiete, Datenquellen und für andere Aspekte RDF Graphen aufgebaut. Zu den Bekanntesten zählen der Thesaurus WordNet für die englische Sprache und DBPedia, der das angesammelte Wissen aus Wikipedia extrahiert und in Beziehung bringt.

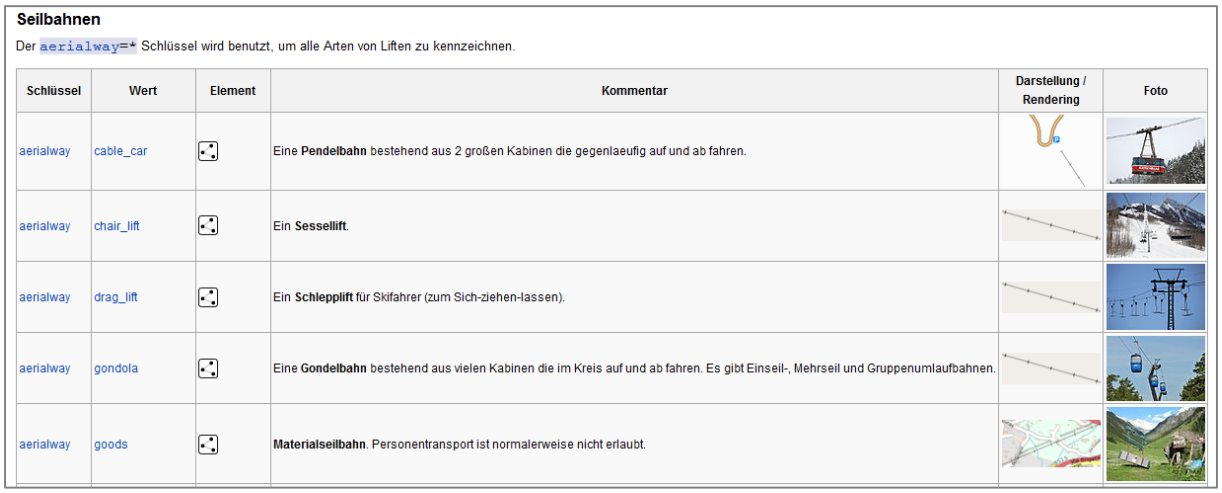
Im Zusammenhang mit OpenStreetMap gibt es für geografische Daten die Semantic Webs GeoNames [9] und LinkedGeoData [10].

3 Aktuelle Tagsuche

In diesem Kapitel werden die bestehenden Möglichkeiten aufgezeigt, die das Auffinden von richtigen OSM Tags erleichtern sollen.

3.1 OSM Wikiseite

Map Features ist eine Seite im OpenStreetMap Wiki [16] und wahrscheinlich aktuell die wichtigste Anlaufstelle um Tags zu suchen. Die Wikiseite ist hauptsächlich eine Zusammenstellung der wichtigsten Tags. Die Tags sind in den 3 Hauptkategorien „Gegenständlich“, „Nicht Gegenständlich“ und „Benennungen“ und darunter in ca. 70 Unterkategorien eingeordnet. Eine Unterkategorie entspricht einem Key. So gibt es zum Beispiel eine Unterkategorie des Keys `aerialway`, in welcher alle möglichen Seilbahnen aufgelistet werden (Siehe Abb. I-6). Eine weitere gute Wikiseite ist *How to Map A* [17]. Im Gegensatz zu *Map Features* - welche eigentlich nach Keys sortiert ist - ist *How to Map A* eine Liste die alphabetisch nach Objekten sortiert ist, die in der realen Welt vorfindet.
Gemeinsam haben diese Seiten, dass für die Suche ein Teil des Tagnamens (Key und Value) bekannt sein muss oder man hat Glück und der Suchbegriff kommt im Kommentarfeld vor.



Seilbahnen

Der `aerialway=*` Schlüssel wird benutzt, um alle Arten von Liften zu kennzeichnen.

Schlüssel	Wert	Element	Kommentar	Darstellung / Rendering	Foto
aerialway	cable_car		Eine **Pendelbahn** bestehend aus 2 großen Kabinen die gegenlaeufig auf und ab fahren.		
aerialway	chair_lift		Ein **Sessellift**.		
aerialway	drag_lift		Ein **Schlepplift** für Skifahrer (zum Sich-ziehen-lassen).		
aerialway	gondola		Eine **Gondelbahn** bestehend aus vielen Kabinen die im Kreis auf und ab fahren. Es gibt Einseil-, Mehrseil und Gruppenumlaufbahnen.		
aerialway	goods		**Materialseilbahn**. Personentransport ist normalerweise nicht erlaubt.		

Abb I-6: Ausschnitt aus Map Features. Gezeigt wird der Abschnitt „Seilbahnen“

3.2 TagInfo

Die TagInfo Website [18] gibt einen Einblick in alle mögliche Statistiken über Tags. Sie hilft dabei den Mappern indem sie Informationen darüber aufbereitet, welche Tags in der OSM-Datenbank vorkommen, wieviele Leute einen bestimmten Tag eingetragen haben und wo diese auf der Karte benutzt wurden, etc. Auch hier müssen für das Auffinden von Tags Teile des Tagnamens bekannt sein, jedoch besteht die Möglichkeit gezielt nach Keys oder Values zu suchen. Die Resultate lassen sich auch nach der Häufigkeit der Tags sortieren.
TagInfo hat auch sehr gute Webservices, die Zugang zu den OSM-Tagdaten bieten. Diese Webservices werden in diese Arbeit extensiv genutzt.

4 Attributive Heterogenität

Eine Daten-Heterogenität bezeichnet eine Uneinheitlichkeit in der Menge der Daten. In OpenStreetMap ist eine Uneinheitlichkeit problematisch. Die attributive Heterogenität und dessen Auswirkungen sind wichtig für diese Arbeit und werden in diesem Kapitel genauer beleuchtet. Neben der attributiven Heterogenität können zwei weitere Arten an Heterogenitäten auftreten:

- Syntaktische Heterogenität:
  Derselbe Sachverhalt wird syntaktisch unterschiedlich dargestellt.
  Zum Beispiel `amenity=bank;atm`, `amenity=atm;bank` oder (`amenity=bank` und `atm=yes`). [11]

- Geometrische Heterogenität:
  Viele Kartenobjekte können entweder als einzelne Punkt oder als Fläche betrachtet werden.
  Zum Beispiel: Eine Linie `natural=tree_row` für eine Alee oder mehrere Bäume als Punkte mit dem Tag `natural=tree`

Die attributive Heterogenität bezieht sich auf die Diversität der Tags, die das gleiche Objekte beschreiben. Grundsätzlich beruht sie auf dem semantischen Verständnis der Tags. Ein Paper von A. Vandecasteele und R. Devillers geht von drei Hauptursachen aus (Abb. I-7). [12]
Im Folgenden wird auf diese Ursachen bezogen, diese können aber noch weiter in Unterkategorien aufgeteilt werden. Zudem kann man mit den Tagging-Schemas einen weiteren Aspekt hinzuzählen.

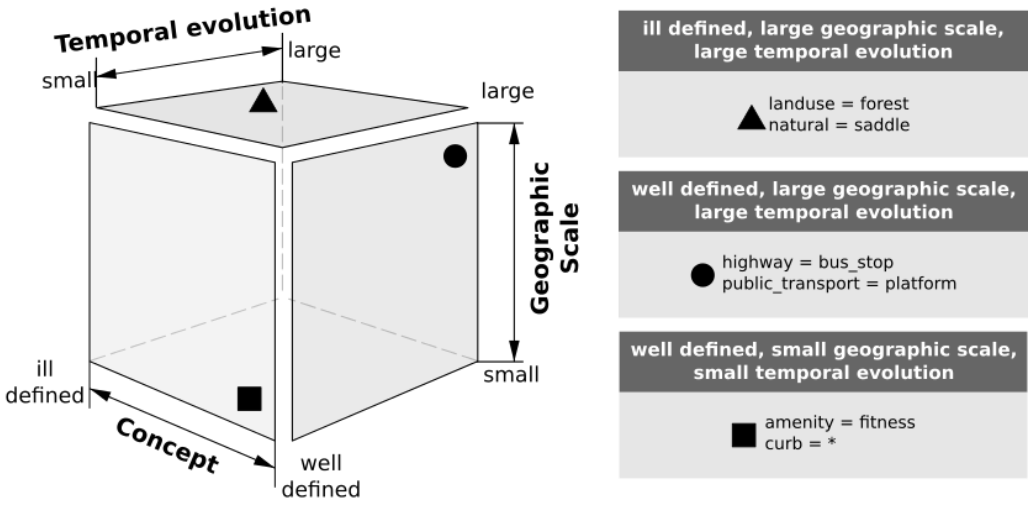


Abb I-7: Übersicht über drei Ursachen attributiver Heterogenität. [12]

Tags und Tagging-Schemas (Siehe Kapitel I-2.2) können häufig mehreren Kategorien zugeordnet werden.

Es gibt folgende Kategorien:

- Mehrdeutige Konzepte
- Zeitliche Entwicklung
- Geografische Diversität
- Tagging-Schemas

Diese werden im Folgenden mit Beispielen erläutert.

4.1 Mehrdeutige Konzepte (Concept)

4.1.1 Unklare Definition und Subjektivität

Es herrscht allgemein keine Übereinkunft wie das Konzept zu definieren ist. Das Konzept ist aber bereits in der Realität schwer abzugrenzen oder subjektiv anders interpretierbar.
Auswirkung: Die Community ist sich nicht einig, welche Tags für ein Konzept verwendet werden sollen. Der Einzelne geht nach seinem eigenen Tagging-Muster und seiner Gewohnheit vor. Schon eingetragene Objekte ändern häufiger ihre Tags.

Beispiele:

- `landuse=forest` oder `natural=wood` [13]
- `amenity=fitness` oder `amenity=fitness_center` oder `amenity=gym` oder `amenity=sports_center` oder `leisure=fitness` oder … [12]
- `natural=saddle` oder `mountain_pass=*` [12]
- `highway=track` oder `tracktype=<1-5>`
- `shop=general` oder `shop=supermarket`
- `amenity=pub` (angenehme Atmosphäre) oder `amenity=bar` (lärmig)

4.1.2 Ähnliche und Synonyme Tags

Konzepte, die sich nur in wenigen Details unterscheiden und deshalb unterschiedliche Tags haben. Synonyme Tags entstehen oft, wenn neue Tags erstellt werden, obwohl für das Konzept bereits ein Tag unter anderem Namen existiert. (Siehe Kapitel I-4.2)
Auswirkung: Es kommt zu weniger korrektem Tagging oder ein Objekt wird gleich mit allen ähnlichen Tags gekennzeichnet.

Beispiele:

- `parking=garage_boxes` oder `landuse=garage` oder `landuse=garages` oder `amenity=parking`
- `waterway=river` oder `waterway=canal`
- `waterway=ditch` oder `waterway=drain` oder `waterway=stream`
- `highway=footway` oder `highway=pedestrian`
- `natural=sand` oder `surface=sand`
- `amenity=grave_yard` oder `landuse=cemetery`
- `leisure=swimming_pool` oder `amenity=swimming_pool`
- `amenity=architect_office` oder `office=architect`
- `leisure=bbq` oder `amenity=bbq`
- `building=supermarket` oder `shop=supermarket`

4.1.3 Überladene Tags

Ein Tag, der zwei (oder mehr) unterschiedliche Konzepte vereint, die eigentlich schon eigene Tags haben.
Auswirkung: Es wird der überladene Tag oder das Tagging Schema verwendet.

Beispiele:

- `leisure=adult_gaming_center` oder (`leisure=amusement_arcade` und `adult=yes`)
- `amenity=beach_resort` oder (`natural=beach` und `tourism=resort`)

4.2 Zeitliche Entwicklung (Temporal Evolution)

Über die Zeit können sich Tags und deren Definition entwickeln, sie werden durch besser beschreibende Tags abgelöst.
Auswirkung: Die alten Tags werden immer weniger gebraucht, als „Depricated Feature“ [14] gekennzeichnet und durch neue, empfohlene Tags ersetzt. Die alten Tags können jedoch vereinzelt noch auftreten.

Beispiele:

- `highway=bus_stop` oder `railway=platform` oder `public_transport=platform` [12]
- `landuse=farm` oder `landuse=farmland` oder `landuse=farmyard`
- `historic=yes` oder `historic=museum` oder `tourism=museum`

4.3 Geographische Diversität (Geographic Scale)

Communities in unterschiedlichen Kulturkreisen entwickeln eigene Tags oder wenden gleiche Tags auf unterschiedlichen Konzepten an.
Auswirkung: Grenzübergreifend oder sogar bei lokalen Untergrenzungen sind die Tags nicht einheitlich.

Beispiele:

- `water=pond` (in Kanda) oder `water=lakes` (in anderen Ländern) [12]
- `sport=football` oder `sport=soccer`
- `landuse=village_green` (in Grossbritanien) oder `leisure=park` (in anderen Ländern)
- `shop=general` (hat eine andere Bedeutung in Grossbritianien als in den USA und Australien)

4.4 Tagging Schemas

4.4.1 Ober- und Unterbegriffe

Zu dieser Kategorie gehören die Tags, die mehrere Konzepte als Gruppe zusammenfassen. Als Haupttag beschreiben sie ein Objekt generell und deren Untertags definieren es genauer.
Auswirkung: Das Objekt wird nur mit einem Haupttag attribuiert. Die Untertags werden teils oder ganz weggelassen. Umgekehrt können die Untertags für ein Objekt auch alleine verwendet werden, wenn sie selbsterklärend sind.

Beispiele:

- `tourism=zoo` ist Oberbegriff von `zoo=petting_zoo`
- `amenity=restaurant` ist Oberbegriff von `amenity=cafe` oder `cuisine=coffee_shop` oder …
- `shop=department_store` ist Oberbegriff von `shop=boutique` oder `shop=clothes` oder `clothes=*`
- `shop=furniture` ist Oberbegriff von `furniture=bedroom` ist Oberbegriff von `shop=bed`
- `amenity=place_of_worship` ist Oberbegriff von `religion=christian` ist Oberbegriff von `denomination=catholic` ist Oberbegriff von `denomination=roman_catholic`
- `landuse=construction` ist Oberbegriff von `construction=residential`
- `landuse=industrial` ist Oberbegriff von `power=plant`, `generator:method=water-storage` ist Oberbegriff von `generator:method=water-pumped-storage` ist Oberbegriff von `waterway=dam`
- `building=stable` ist Oberbegriff von (`building=yes` und `building:use=stable`)
- `leisure=nature_reserve` ist Oberbegriff von `boundary=national_park` oder (`boundary=protected_area` und `protect_class=2`)

4.4.2 Zusätzliche Informationen

Einem Objekt kann mit zusätzlichen Tags mehr Informationen und Details gegeben werden. Die Kombination der Tags um ein Konzept zu beschreiben kann unterschiedlich sein. Empfohlene Tagging-Schemas können vom Editor vorgeschlagen werden.
Auswirkung: Die Mapper verwenden ihre eigenen Tagging-Schemas.

Beispiele:

- (`highway=cycleway` und `foot=designated`) oder (`highway=footway` und `bycicle=designated`) oder (`highway=path` und `bycicle=designated` und `foot=designated`) [15]
- `leisure=firepit` oder (`tourism=picnic_site` und `fireplace=yes`)
- (`amenity=parking` und `capacity=*`) oder `amenity=parking_space`

5 Vorgehen

Für die Planung des Projekts TagFinder wurde das Vorgehensmodell RUP (Rational Unified Prozess) gewählt, welche eine inkrementelle, iterative Softwareentwicklung erlaubt.
Zunächst musste die zu Verfügung stehende Projektzeit in Phasen aufgeteilt und eine Grobplanung für die Arbeitspackete erstellt werden. Bei der Planung musste eine längere Einarbeitungszeit berücksichtigt werden, da erst die Programmiersprache Python erlernt und grundlegendes Wissen um Begriffsnetzwerke angeeignet werden musste.
Anschliessend ging es daran, so schnell als möglich einen einfachen Thesaurus zu erstellen, da er die Basis der Suchmaschine darstellt. Mit Hilfe der Webservices von TagInfo konnte das Vorhaben zügig umgesetzt werden und ein erster Prototyp war vorzeigefähig.
In der nächsten Phase konnte der Thesaurus um die wichtigen Begriffsbeziehungen erweitert werden. Leider war es nur möglich dies manuell zu bewerkstelligen, da im Bereich der Begriffsnetzwerke auf wenig zurückgegriffen werden konnte. So musste jeder Key und Tag im Basis-Thesaurus einzeln betrachtet und Synonyme, Ober- und Unterbegriffe in Deutsch und Englisch hinzugefügt werden. Zur erleichterten Erfassung wurde ein simples Kommandozeilen-Tool entwickelt, dass mit Hilfe von Online-Wörterbüchern wie OpenThesaurus oder Wordnik Vorschläge zu den Begriffen heraussucht. Dennoch, die Bearbeitung des Thesaurus war sehr aufwenig und nahm viel Zeit in Anspruch: Es wurde schnell klar, dass bis Ende des Projekts nicht alle Tags durchgearbeitet werden konnte und die Priorität musste auf den Information Retrieval Prozess, den Suchalgorithmus und die restlichen Anforderungen, wie Webservices und Website-Design gesetzt werden.

6 Ergebnisse

6.1 Zielerreichung

Die Arbeit „Tag-Suchmaschine und Thesaurus für OpenStreetMap“ konnte am Schluss die gesetzten Ziele erreichen. Vorallem die TagFinder Webseite ist sehr gelungen und es konnten sogar einige optionale Features implementiert werden. Nicht ganz vollständig muss jedoch der TagFinder Thesaurus abgegeben werden. Aufgrund des aufwendigen Erstell- und Bearbeitungsprozesses konnte nur die wichtigsten Keys und Tags vollumfänglich bearbeitet und um verwandte Begriffe, Ober- und Unterbegriffe erweitert werden. Jedoch sind auch die unbearbeiteten Tags besser auffindbar als bei den existierenden Tag-Suchen. Bei Tests (Siehe Kapitel II-4.1.2) konnte gezeigt werden, das die Suchmaschine fähig ist, für nahzu alle Begriffe eines Editor Presets die passenden Tags zu finden.

6.2 Ergebnisse der TagFinder - Webseite

- Funktionsfähige Suchmaschine für OpenStreetMap Tags. Deutsche Suchbegriffe werden zusätzlich übersetzt und Rechtschreibfehler werden beachtet.
- Die gefundenen Tags werden nach Haupt-, Ober- und Unterbegriffen sortiert und mit folgenden Information ausgegeben:
  - Tagname mit Links zur OSM-Wikiseite des Keys und des Tags
  - Eine Beschreibung des Tags.
  - Listen an anderen Tags, die mit dem gefunden Tag impliziert, kombiniert oder auf dessen Wikiseite verlinkt werden.
  - Eine Liste an verwandten Begriffen.
  - Informationen zur Suche: Was wurde gesucht? Wieso wurde der Tag gefunden? Zudem wird der Suchbegriff „highlighted“.
  - Statistiken des Tags: Totale Anzahl in OSM, Anzahl Vorkommnisse als Punkte, Relationen, Linien und Flächen.
  - Hinweise auf welche Datenelemente der Tag angewendet werden kann.
  - Falls vorhanden, das Bild aus der Wikiseite.

- Die Webseite ist komplett auf Deutsch und Englisch.
- Im Suchfeld wird während der Eingabe Vorschläge angezeigt („Meinten Sie…?“)
- Das Design der Webseite ist einheitlich, intuitiv und dank Bootstrap auch responsive: Es kann auf vielen Smartphone-Displays gesucht und die Inhalte angezeigt werden.
- Die Webseite unterstützt OpenSearch

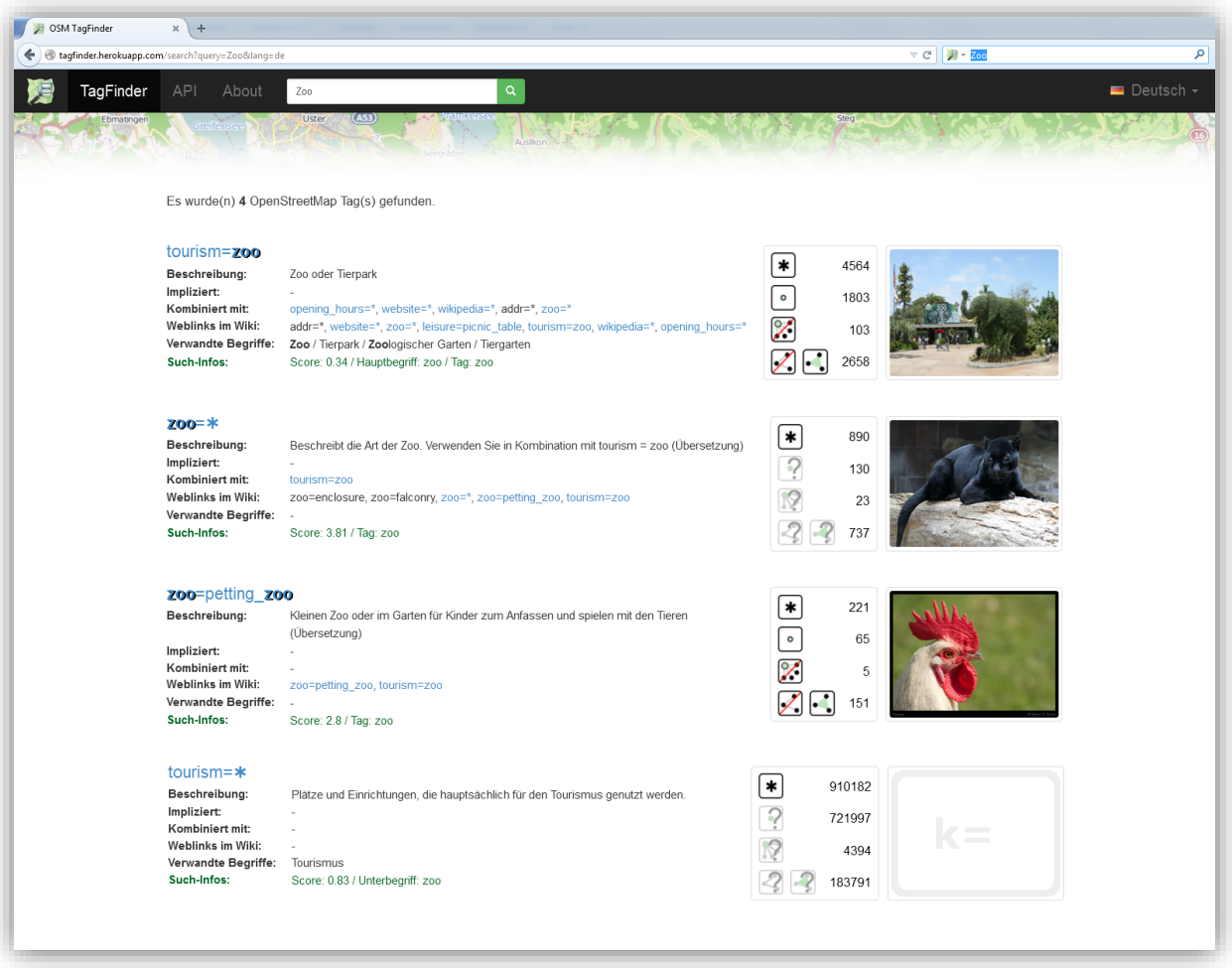


Abb I-8: TagFinder Ausgabe bei der Suche nach „Zoo“

6.3 Ergebnisse der TagFinder - Webservices

- Es werden vier Webservices angeboten. Die Rückgabe erfolgt im JSON Format:
  - api/search: Liefert dieselben Resultate wie eine Suche über die Webseite.
  - api/tag: Rückgabe aller verfügbaren Informationen für einen bekannten Tag.
  - api/terms: Liefert zu einem Begriff alle verwandten Begriffe, sowie Ober- und Unterbegriffe, in Deutsch und Englisch.
  - api/suggest: Bietet Vorschläge zu einem Suchbegriff an, in Englisch oder Deutsch.

- Auf der Webseite kann die komplette API Dokumentation mit Erläuterungen und Beispielen angesehen werden (oder im Kapitel IV-6).
- Unterstützung von JSONP und CORS für clientseitige Anfragen.
- Für bessere Lesbarkeit können die Resultate optional mit einer Einrückung zurückgegeben werden.

6.4 Ergebnisse des TagFinder - Thesaurus

- Der Thesaurus enthält einerseits OpenStreetMap Keys und Tags, sowie einige Taginformationen (wie z.B. die Beschreibung und Statistiken), andererseits aus „Begriffs-Konzepten“. Die Konzepte werden mit den SKOS Relationen miteinander verbunden und bilden ein komplexes Begriffsnetzwerk.
- Der TagFinder Thesaurus kann im RDF/XML Format von der Webseite heruntergeladen werden.
- Der RDF Graph beinhaltet 2574 Keys und Tags (776 Key, 1798 Tags). Davon wurden beinahe 800 bearbeitet und um die Begriffe erweitert.
- Insgesamt hat der Thesaurus 53754 Triple.
- Enthält ein Mapping in den „OSM Semantic Network“ Thesaurus [19].
- Der Thesaurus kann automatisch oder manuell aktualisiert werden. Erneuert wird alles ausser die Begriffskonzepte und dessen Beziehungen.
- Es wurde ein Kommandozeilen-Tool entworfen, mit dem ein neuer TagFinder Thesaurus angelegt oder ein bereits bestehender Thesaurus weiterbearbeitet werden kann.

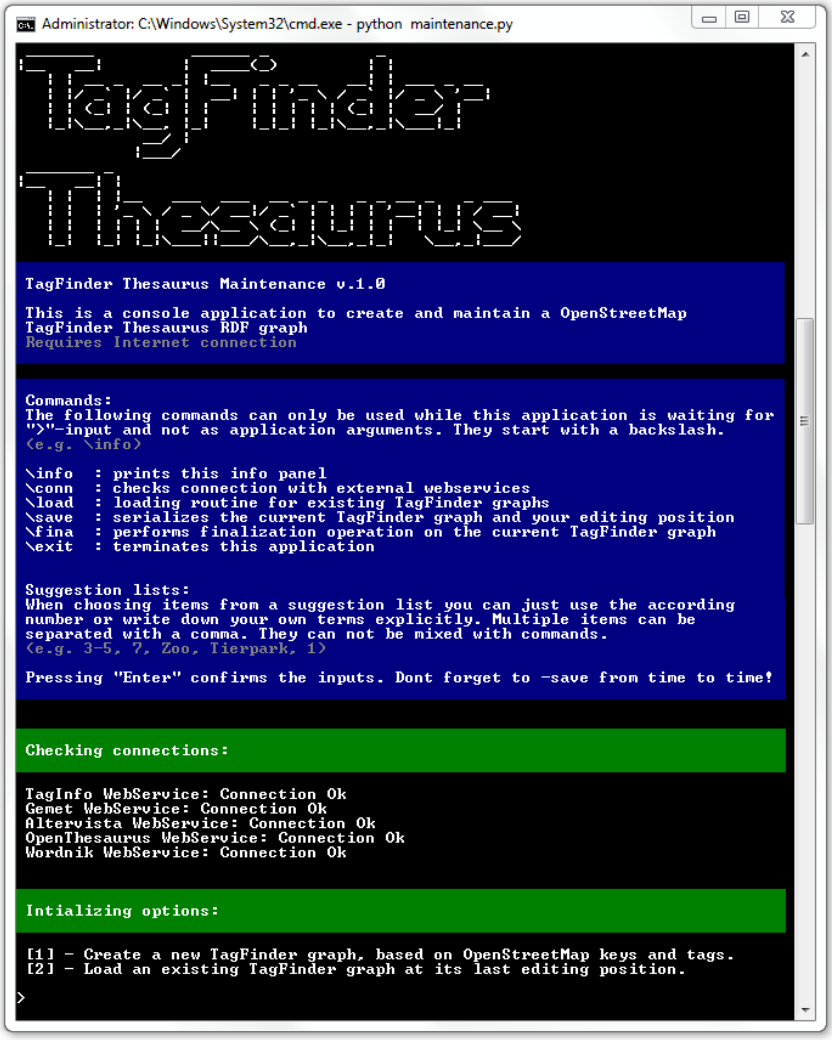


Abb. I-9: Das Kommandozeilen-Tool für die Bearbeitung des TagFinder Thesaurus

6.5 Ausblick

Im Rahmen dieses Projekts konnte ich mir viele neue Fähigkeiten aneignen. Dazu gehört neben Python auch der Umgang mit modernen Webtechnologien. Die Beschäftigung mit den Begriffsnetzwerken war sehr spannend, haben diese doch viel mit künstlicher Intelligenz zu tun.

Die TagFinder Webseite und die Suchmaschine stehen gut da. So habe ich bereits positive Feedbacks aus der OpenStreetMap Community erhalten und er wurde sogar im deutschsprachigen OSM Wochenblog erwähnt. Dennoch sehe ich einigen Raum für Weiterentwicklungen. Ein nützliches Feature wäre das Hinzufügen einer Sortierfunktion, die erlaubt alle Resultate nach Trefferscore oder komplett nach Statistiken der Tags zu sortieren. Weitere Verbesserungen wären im Bereich des Suchprozesses und der Vorschläge möglich: So hat der TagFinder z.B. noch Schwierigkeiten im Umgang mit Suchbegriffen im Plural.

Beim Thesaurus sollten die restlichen Tags und Keys nachgeführt werden. Ist dieser Schritt getan, lässt sich mit dem Netzwerk auch komplexe Analysen der Tags und Begriffen anstellen. Auch eine Homogenisierung der OSM-Daten wäre dann denkbar.

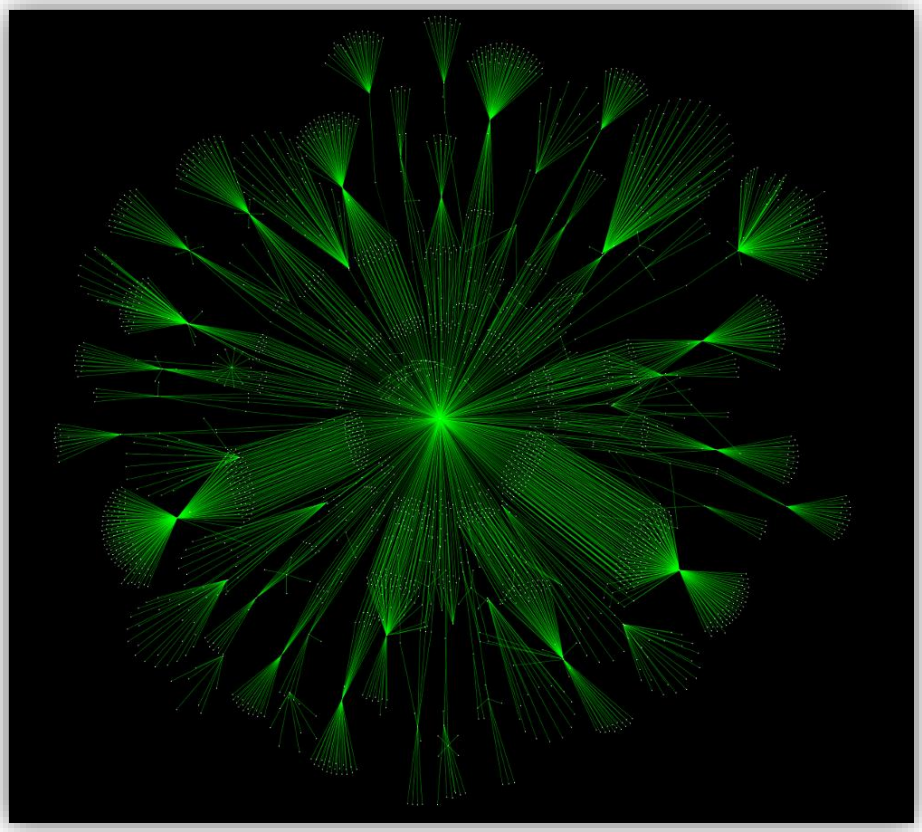


Abbildung: Übersicht über den TagFinder Thesaurus Graph. Punkte stellen OSM-Tags / -Keys oder Begriffe dazu dar. Verbunden werden sie mit hierarchischen Relationen.
Im Mittelpunkt steht das Konzept „RelatedTerm“. (Der Graph wurde mit Cytoscape 3.2.0 berechnet)

7 Persönlicher Bericht

Die Umsetzung der Studienarbeit war das bislang grösste Projekt, dass ich in Angriff nahm. Zudem sah ich mich mit vielen neuen Themen konfrontiert. So brachte ich beispielsweise keine Erfahrung im Bereich der Webentwicklung mit. Auch die Programmiersprache Python war komplett neu für mich, bis anhin kannte ich nur „statically typed“ Sprachen wie Java oder C++.
Die Einarbeitung in diese Themen erhöhte auf der einen Seite zwar den Aufwand, steigerte auf der anderen Seite aber auch massiv den Lernerfolg. Heute kann ich sagen, dass ich fähig bin eine Webseite zu enwickeln.
Desweiteren habe ich einiges beim Projektmanagement dazugelernt: Ein ständiger Kontextwechsel – zum Beispiel zwischen Website Design und Server-Programmierung – kann unglaublich zeitraubend sein. Falls möglich sollte ein unabhängiger Projektteil zuerst abgeschlossen werden, bevor ein anderer begonnen wird und sollte in die Planung einfliessen. Beim Zeitmanagement hat mir dieses Projekt geholfen meine eigenen Fähigkeiten und wie lange ich für die Umsetzung brauche besser abzuschätzen. Von nun an muss ich für alle Einzeltätigkeiten und vor allem für die Schlussphase mehr Zeit einrechnen.
Zum Schluss kann ich sagen, dass diese Arbeit sehr intensiv war, aber auch viel Spass gemacht hat.

8 Danksagung

An dieser Stelle möchte ich mich ganz speziell bei meinem Betreuer Prof. Stefan Keller bedanken. Ohne seine kompetente und motivierende Hilfestellung wäre die TagFinder Applikation nicht in dieser Form zustande gekommen. Bei den Reviews konnten stets wichtige Entscheidungen besprochen werden und gute Verbesserung zu Dokumenten und der Anwendung wurden vorgeschlagen, auch wenn die Meetings dann länger dauerten als geplant.

Teil II: SW-Projektdokumentation

1 Anforderungsspezifikation

1.1 Einleitung

In diesem Kapitel werden die Anforderungen der Benutzer an das System erfasst.
Die Aufgabenstellung (Siehe Kapitel I-1.1) geht von drei Teilsystemen aus, die im Folgenden getrennt auf ihre Aktoren und Use Cases geprüft werden.

1.2 Aktoren, Use Cases und User Szenarios

1.2.1 TagFinder - Webseite

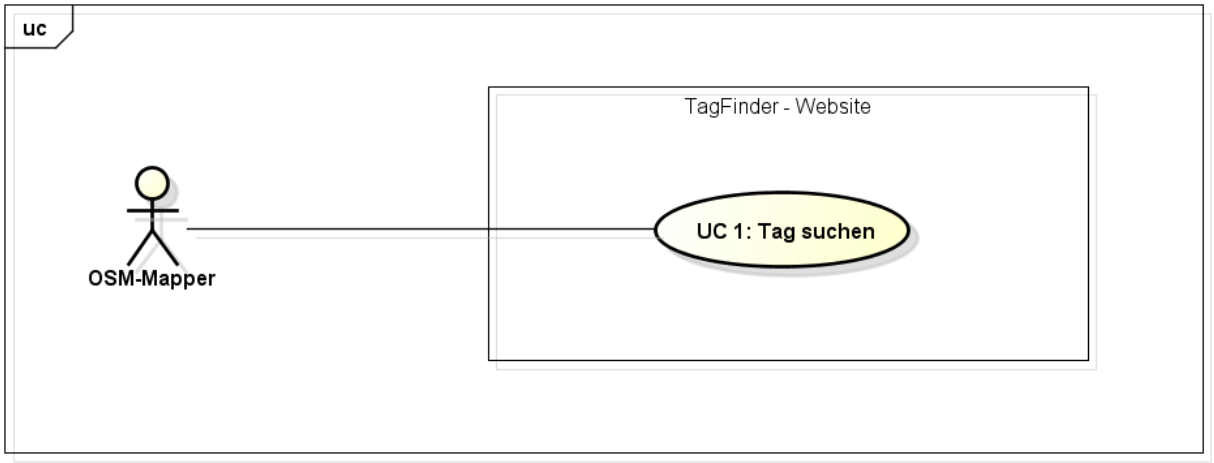


Abb II-1: Use Case Diagramm für die TagFinder - Webseite

Aktoren: OSM-Mapper

OSM-Mapper arbeiten freiwillig und aktiv an OpenStreetMap mit. Ausgerüstet mit GPS-Empfängern durchwandern sie die Umgebung und notieren interessante Kartenobjekte. Zuhause tragen sie diese in der Karte ein und bezeichnen den Objekttyp mit einem oder mehreren Tags. Anschliessend veröffentlichen sie ihre neuen Daten Online.
OSM-Mapper haben grundsätzliche Kenntnisse in der Bedienung von Computern.
Zurzeit gibt es mehr als 1.9 Millionen OpenStreetMap User (Siehe Kapitel I-2.1).

Use Cases:

UC 1: Tag suchen
Der OSM-Mapper gibt bei der TagFinder - Webseite einen Suchbegriff ein und erhält eine sortierte Resultatliste zurück, welche alle relevanten Tags beinhaltet. Diese sind so präsentiert, dass alle benötigten Informationen klar ersichtlich sind.

1.2.2 TagFinder - Webservice

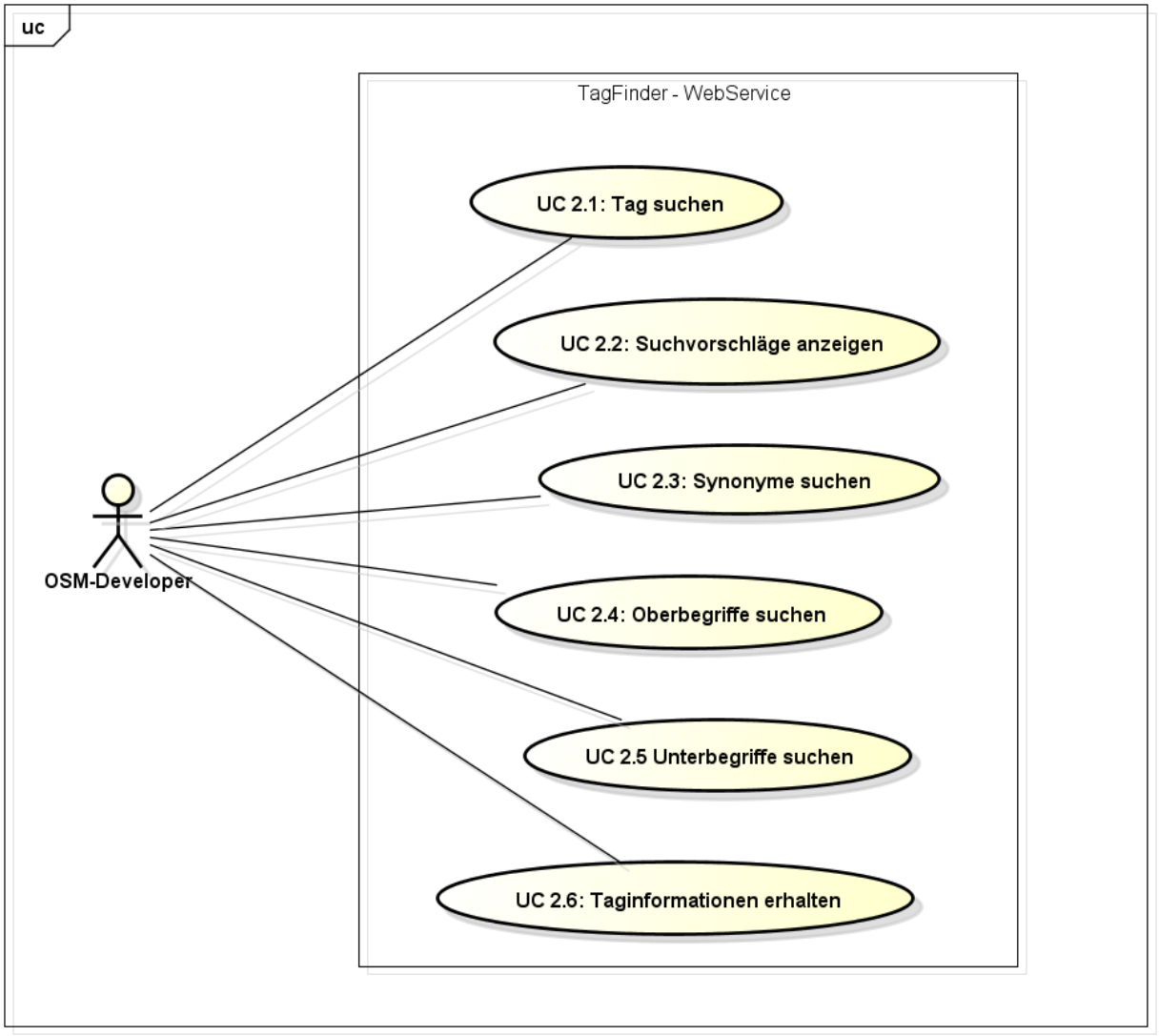


Abb II-2: Use Case Diagramm für den TagFinder - Webservice

Aktoren: OSM-Developer

Die OSM-Developer entwickeln Software, die auf OpenStreetMap basiert. Es kann sich dabei um ein Tool oder Plugin handeln, dass OSM-Mappern hilft Daten zu erfassen oder sie entwerfen eine Applikation die für verschiedenste Benutzer gedacht ist, wie Navigationshilfen, Routenplaner, Augmented Reality Spiele etc.
OSM-Developer haben als Software-Entwickler weit fortgeschrittene Computerkenntnisse. Grundsätzlich erwarten sie vom Webservice ein gutes API Design.

Use Cases:

UC 2.1: Tag suchen
Dieser Use Case deckt sich mit UC 1, hat aber den Unterschied, dass Anfragen statt über die Webseite über den Webservice laufen und die Resultate in einer maschinenverständlichen Form ausgegeben werden.

UC 2.2: Suchvorschläge anzeigen
Der Benutzer erhält eine Liste an Suchvorschlägen für einen beliebigen Begriff. Die Liste enthält ähnliche Wortbedeutungen oder Fehlerkorrekturen.

UC 2.3: Synonyme suchen
Der Benutzer kann alle Synonyme zu einem beliebigen Begriff abfragen.

UC 2.4: Oberbegriffe suchen
Der Benutzer kann alle Oberbegriffe zu einem beliebigen Begriff abfragen.

UC 2.5 Unterbegriffe suchen
Der Benutzer kann alle Unterbegriffe zu einem beliebigen Begriff abfragen

UC 2.6: Taginformationen erhalten
OSM-Mapper können eine Liste an relevanten Informationen zu den Tags oder Keys anfordern. Dazu gehören die OSM-Wikiseite, eine Beschreibung, einen Link zum Anzeigebild, Synonyme und Ober- und Unterbegriffe.

1.2.3 TagFinder - Thesaurus

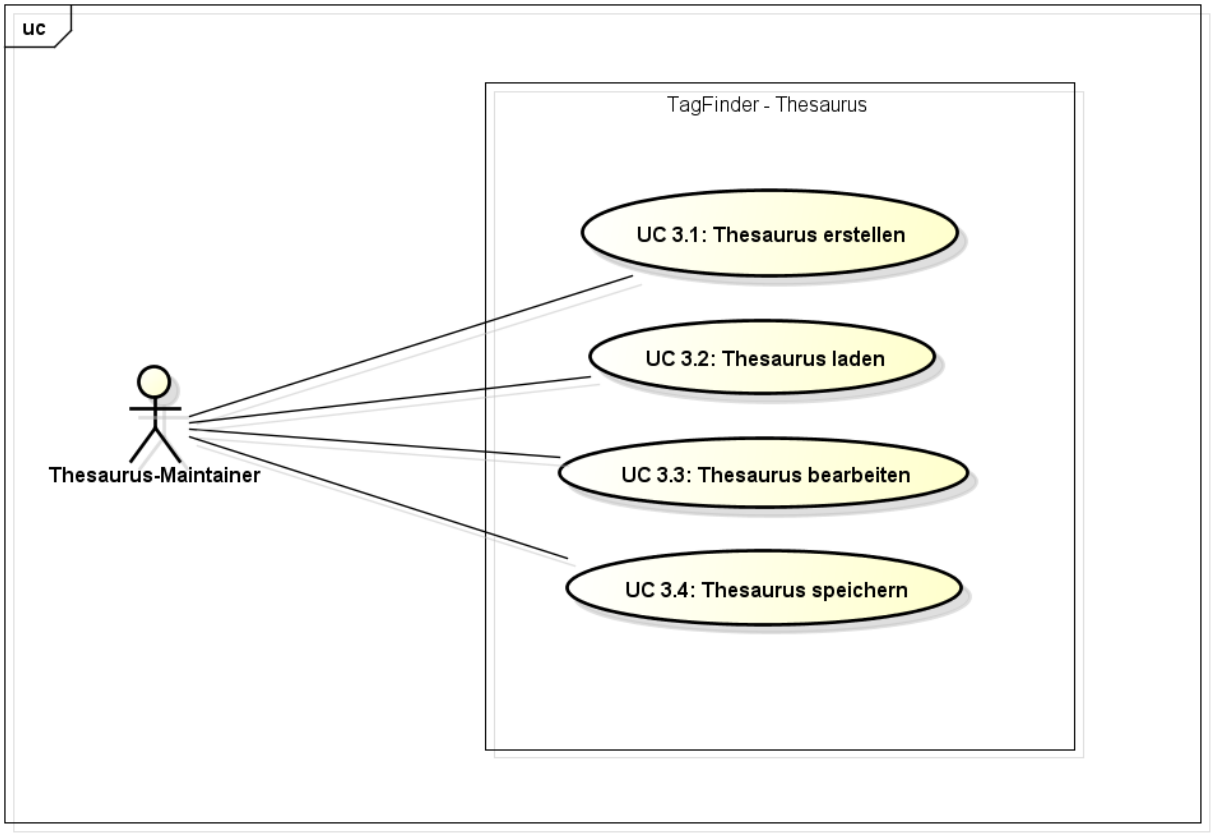


Abb II-3: Use Case Diagramm für den TagFinder - Thesaurus

Aktoren: Thesaurus-Maintainer

Die Maintainer kümmern sich um die Wartung des TagFinder Systems. Sie sind auch daran interessiert, den Thesaurus aktuell zu halten. Dabei verlangen sie, dass der Aktualisierungsprozess den laufenden Betrieb nicht unterbricht.

Thesaurus-Maintainer haben neben langer Computererfahrung auch ein tieferes Verständnis des TagFinder Systems.

Use Cases:

UC 3.1: Thesaurus erstellen

Der Benutzer kann einen neuen Thesaurus anlegen. Als Basis dienen die OpenStreetMap Tags. Der Thesaurus-Maintainer beginnt mit der Bearbeitung (UC 3.3).

UC 3.2: Thesaurus laden

Ein bestehender Thesaurus wird in die Applikation geladen. Der Benutzer kann mit der Bearbeitung am gewünschten Ort fortfahren (UC 3.3).

UC 3.3: Thesaurus bearbeiten

Der Benutzer kann den Thesaurus um Synonyme, Ober- und Unterbegriffe erweitern und manuell Änderungen an diesen und den Taginformationen vornehmen. Er wird dabei mit Vorschlägen unterstützt.

UC 3.4: Thesaurus speichern

Da das Hinzufügen von Semantik und Einordnen der Begriffe manuell erfolgen muss, kann die Bearbeitung einige Zeit dauern. Aus diesem Grund soll der Maintainer die Begriffssammlung jederzeit zwischenspeichern können. Dieser Use Cases gilt dann auch für die Fertigstellung des Thesaurus.

2 Analyse und Design

2.1 Domain Model

Der TagFinder hat leider keine wirkliche Business Logik. Was der Domain am nächsten kommt, ist die logische Struktur des Thesaurus. Problem bei RDF ist allerdings die fehlende Trennung von Objektinstanz und Schema.

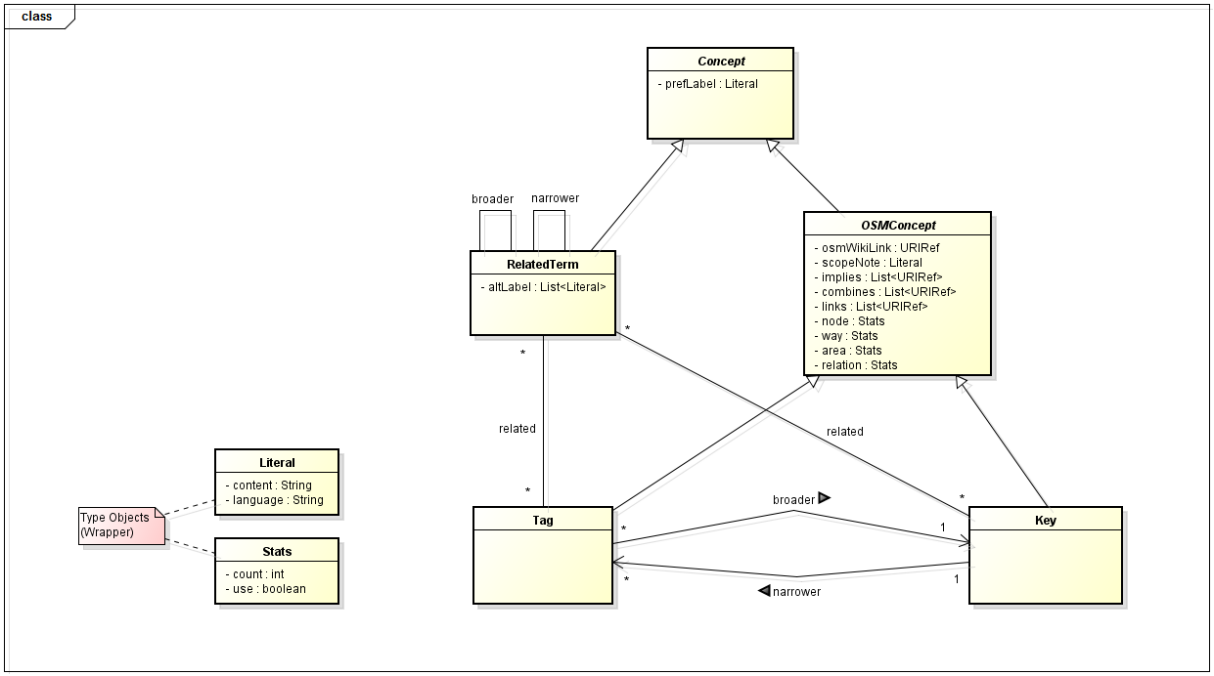


Abb II-4: „Klassendiagramm“ des TagFinder Thesaurus

Ansonsten zählen die Klassen im Package „search“ zum Application Logic Layer. Diese sind jedoch nicht sehr interessant:

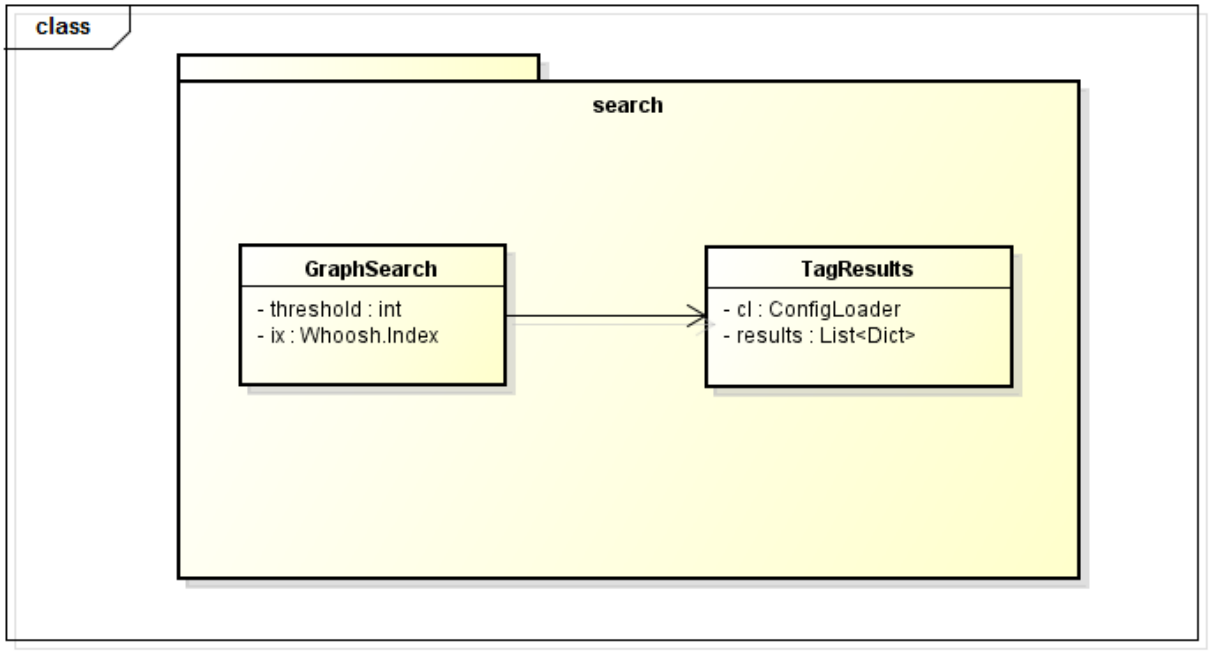


Abb II-5: Klassendiagramm, Package „search“

2.2 Architektonische Ziele und Einschränkungen

Die architektonischen Hauptziele im Projekt TagFinder sind:

- Austauschbare Packages
- Einfache Erweiterbarkeit
- Einfache Portierung auf ein neues Betriebssystem

2.2.1 Austauschbare Packages

Die Pakete werden in logische Einheiten aufgeteilt und erfüllen ihre zentrale Aufgaben. Zum Beispiel sollen Aufgaben, die Netzwerkanfragen beinhalten, komplett von den Klassen im Resource Layer getrennt werden.

2.2.2 Einfache Erweiterbarkeit

Um auf Dauer den ändernden Verhältnissen von OpenStreetMap gerecht zu werden, ist es notwendig das TagFinder System anpassen und erweitern zu können.

2.2.3 Einfache Portierung auf ein neues Betriebssystem

Durch die Anwendung von Python als Programmiersprache ist schon der wichtigste Grundstein für die einfache Portierung gelegt: Python ist eine interpretierte Sprache und die Anwendung kann im Quellcode ausgeliefert werden. Dieser Punkt stellt jedoch eine Einschränkung für die externen Libraries dar. Nur reine „pythonic“ Bibliotheken kommen in Frage (d.h. keine C-Libraries / Systemaufrufe).

2.3 Distributed Presentation Architecture

Der TagFinder hat eine 2-Tier Client-Server Architektur. Der Serverteil ist für 3 bzw. 4 Layers zuständig (Siehe Abb. II-6):

- Resource Layer: Der Resource oder Data Access Layer beinhaltet die ganze Datenhaltung und übernimmt das Laden und Speichern (bzw. Serialisieren).
- Application Logic Layer: Nimmt Anfragen aus der höheren Schicht entgegen, berechnet aus den Daten des Resource Layers ein Resultat und meldet diese zurück.
- Presentation Logic Layer : Ist im Webbereich für Rendering der HTML Seite zuständig.
- Im Fall von TagFinder ist ein Service Layer vorgeschoben, welche das Routing und bereitstellen von Webresourcen übernimmt. Falls nötig werden Parameter aus Anfragen an die tiefere Schicht weitergeleitet.

Ein Teil der Präsentation geschieht auch auf der Clientseite im Browser. Per JavaScript werden Response und HTML Seite angepasst oder AJAX Requests ausgelöst.
Man spricht auch von Distributed Presentation Architecture. [20]

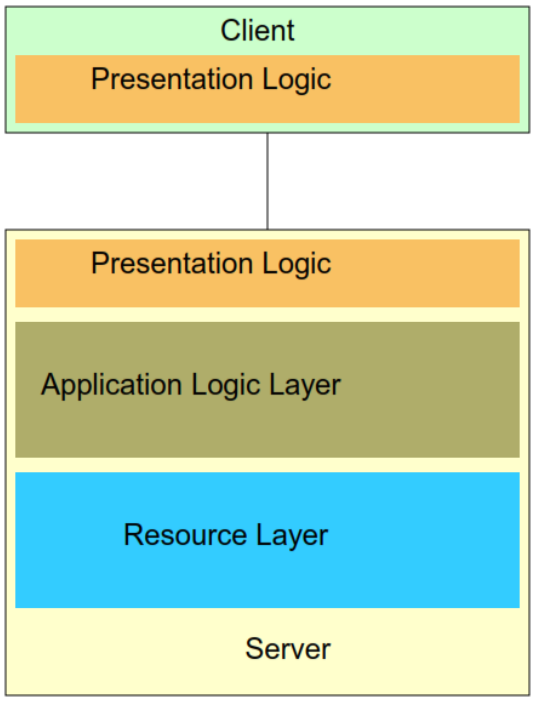


Abb. II-6: Distributed Presentation Architecture [20]

2.4 Packages und Klassen

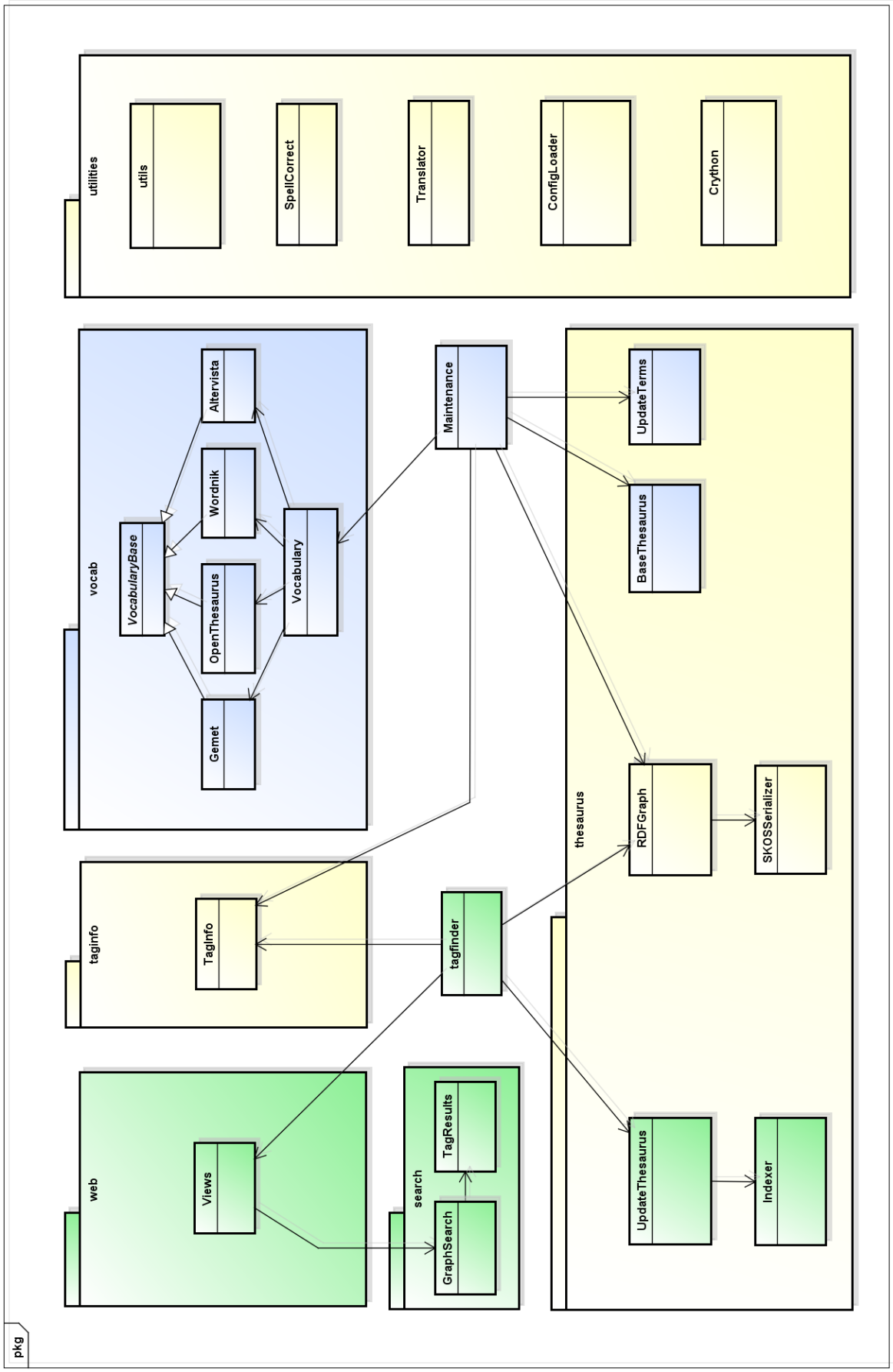


Abb II-7: TagFinder Packagediagramm

In Abbildung II-7 sieht man eine Übersicht über die Architektur in Form eines Packagediagramms und die wichtigsten Klassen. Das Diagramm widerspiegelt nur bedingt die Layeranordnung. Es werden beide Teilsysteme gezeigt, obwohl sie zur Laufzeit unabhängig sind. Packages und Klassen sind mit entsprechender Farbe gekennzeichnet: TagFinder - Webseite in grün und TagFinder - Thesaurus in blau.
Module, die aus praktischen Gründen von beiden Systemen benutzt werden, haben einen gelben Hintergrund.
Die beiden Module Server und Maintenance sind die Startroutinen und befinden sich im root-Verzeichnis.
(Für eine Liste der verwendeten, externen Libraries siehe Kapitel IV-5)

2.4.1 Utilities Package

Dieses Package beinhaltet eine Reihe an Hilfsklassen und wird von fast allen anderen Klassen verwendet. Es werden keine Beziehungen gezeigt, da das Diagramm sonst unübersichtlich würde. Damit keine zirkuläre Beziehungen enstehen wurde darauf geachtet, dass im Utilities keine anderen Klassen aufgerufen werden.

Utils (Modul)
Ist keine Klasse sondern ein Python-Modul, dass einige statische Funktionen zur Umrechnung, Konvertierung und sonstige Hilfsmethoden enthält.

SpellCorrect
Die Klasse implementiert Funktionen die ein String Parameter entgegennehmen und eine Liste an passenden Vorschlägen aus dem Thesaurus retournieren. Kommt bei der Rechtschreibprüfung (Typeahead) und bei der Suche zum Einsatz. Sie greifen auf die Indexer-Files zurück und verwenden die Library „whoosh“. Eine Alternative mit dem Framework „pyenchant“ musste verworfen werden, weil diese eine Installation der C-Library Enchant bedingt.

Translator
Das Übersetzermodul von TagFinder. Englische und deutsche Begriffe können in die andere Sprache übersetzt werden. Ist die Sprache nicht schon vorher bekannt, gibt es eine Funktion, die die Sprache herausfindet und das Wort übersetzt. Die Klasse verwendet „goslate“, eine Python Google Translate API.

ConfigLoader
Lädt und speichert Variablen aus dem Config-File. Die Klasse ist in Package-Sektionen unterteilt.

Crython

Crython ist ein einfacher Crontab-Scheduler, der als Dekorator verwendet wird. Dadurch, dass in der Crython-Bibliothek ein Bug gefixed werden musste, wurde das Modul ins utilities gezogen. Eine Alternative zu Crython wäre der APScheduler (Advanced Python Scheduler), der aber nicht leichtgewichtig ist.

2.4.2 TagInfo

Die Klasse im Package TagInfo kapselt die Anfragen an den Webservice Dienst von TagInfo.org [18]. Wird einerseits von Maintenance für die Erstellung des Thesaurus, andererseits von UpdateThesaurus für die Aktualisierung des Graphen genutzt.

2.4.3 Server (Modul)

Ist das Script, dass den Application Server der Webseite startet. Hier wird auch der Crython-Scheduler angeworfen, der den Thesaurus RDF Graphen aktuell hält. Aus diesem Grund wird der Thesaurus auch im server.py geladen (und nicht in der GraphSearch Klasse) und an den Webserver, wie auch den Scheduler übergeben. Eine Änderung des RDF Graphen durch den Scheduler kann so dem Server direkt bekannt gemacht werden:

```
cl = ConfigLoader()
@crython.job(expr=cl.getWebsiteString('UPDATE_CRONTAB'))
def updateJob():
    print 'updating job started'
    cl = ConfigLoader()
    outputName = cl.getThesaurusString('OUTPUT_NAME')
    outputEnding = cl.getThesaurusString('DEFAULT_FORMAT')
    rdfGraph = RDFGraph(utils.outputFile(utils.dataDir(), outputName, outputEnding, useDateEnding=False))
    tagInfo = TagInfo()
    UpdateThesaurus(tagInfo=tagInfo, rdfGraph=rdfGraph, console=None) # will update the rdfGraph
    today = str(datetime.date.today().strftime("%d.%m.%y"))
    cl.setWebsiteString('DATA_DATE', today)
    cl.write() # storing the date change in config file
    runFlaskApp(rdfGraph, today)
```

Abb II-8: Code Snippet aus dem TagFinder Module. Es handelt sich um den Aktualisiermechanismus des Thesaurus RDF Graphen. Die Methode wird über den Crython-Dekorator in definierten Zeiten aufgerufen.

2.4.4 UpdateThesaurus

Diese Klasse koordiniert die Aktualisierung des Thesaurus. Er kann vom Scheduler instanziert werden (Siehe Abb. II-8) oder von einer einfachen Main-Klasse im root-Verzeichnis (updater.py). Der zweite Fall ist für manuelle Updates gedacht. Als Parameter erhält er einen RDFGraph und eine TagInfo Instanz. Ist das eigentliche Update abgeschlossen, muss der Thesaurus noch indexiert werden.

2.4.5 Indexer

Übernimmt die Aufgabe der eigentlichen Indexierung des Thesaurus, damit in ihm gesucht werden kann. Der Indexer verwendet – wie GraphSearch – die Library „whoosh“ für diese Zwecke: Der Indexer iteriert über alle Triples des Semantik Webs und ordnet die Objekte in definierte Felder ein. Am Ende des Prozesses wird im Ordner ./data/indexer/ „whoosh“-spezifische Files angelegt, die der GraphSearch später bezieht.

2.4.6 GraphSearch und TagResults

GraphSearch ist wohl eine der wichtigsten Klassen und verwendet auch „whoosh“. Hier findet die eigentliche Suche im Thesaurus bzw. dessen indexierten Informationen statt. Eine genauere Beschreibung des Suchprozesses kann im Kapitel II-3 nachgelesen werden.
Auch wichtig ist die Klasse TagResults, welche die gefundenen „Rohresultate“ des GraphSearch weiterverarbeitet: Sie werden sortiert und in eine Form gebracht, mit der die Webseite und Webservice-Routinen etwas anfangen können.

2.4.7 Views

Das Web Package besitzt nur eine zentrale Klasse Views, welche sich um das Routing der Netzwerkanfragen kümmert. Einkommende Requests der Webseite kommen hier an, die Parameter werden ausgelesen und per entsprechender Funktion an die tiefere Schicht weitergeleitet. Berechnete Inhalte werden danach als HTML gerendert oder als JSON zurückgeschickt.

```
@app.route('/search', methods = ['GET'])
def search():
    query = request.args.get('query', '')
    lang = request.args.get('lang', '')
    searchResults = searchCall(query, lang)
    if searchResults is None:
        return redirect('/')

    return render_template('search.html', lang=getLocale(), query=query, results=searchResults)
```

Abb II-9: Code Snippet aus dem Views Module. Gezeigt wird die Funktion, die einkommende „/search“-Anfragen verarbeitet.

Im Views Ordner befinden sich auch die HTML Templates, JavaScript’s, CSS und andere Ressourcen.
Es wird das Python-Microframework Flask verwendet. Dazu muss global eine Flask „app“ instanziert werden. Flask basiert auf dem „Werkzeug-Server“ und kann auf allen Platformen betrieben werden die ein Web Server Gateway Interface (WSGI) unterstützen.
Flask wurde wegen einer Empfehlung als Framework ausgewählt. Alternativen sind zum Beispiel „web.py“ und „Django“.

2.4.8 RDFGraph und SKOSSerializer

RDFGraph ist ebenfalls eine wichtige Klasse und entspricht in etwa einem TabelDataGateway im Datenbankbereich. Die Klasse implementiert hauptsächlich Funktionen für die Extraktion gewünschter Informationen aus dem Thesaurus: Im Grunde ist sie ein Wrapper um die hervorragende Library „rdflib“, welche mit Triple – also mit Subjekt, Prädikat und Objekt – arbeitet. Mit dem Subjekt kann das Konzept im Graphen identifiziert werden. Will man nun zum Beispiel einen verwandten Begriffe einem bekannten Subjekt hinzufügen, kann man die Funktion „addAltLabel(…)“ verwenden (Siehe Abb. II-10). Es gibt dem Beispiel entsprechend viele dieser Wrapper-Funktionen.

```
def addAltLabel(self, subject, obj, language):
    self.graph.add( (URIRef(self.prepareURIRef(subject)),
                     SKOS.altLabel,
                     Literal(self.prepareLiteral(obj), lang=language))
                   )
    return subject
```

Abb II-10: Code Snippet aus der RDFGraph Klasse. Die dargestellte Funktion fügt einem Konzept des Thesaurus ein verwandter Begriff hinzu.

Eine weitere Aufgabe von RDFGraph ist das Laden und Serialisieren eines Thesaurus. Da es jedoch mit „rdflib“ Probleme bei der Serialisation gab, musste der angebotene PrettyXMLSerializer in der eigenen Klasse SKOSSerializer verbessert und dann als Plugin in „rdflib“ eingefügt werden.

2.4.9 Maintenance

Maintenance startet ein einfaches Kommandozeilen-Tool für die Bearbeitung der Synonyme, Ober- und Unterbegriffe. Er bietet die Möglichkeit entweder einen neuen „Basis-Thesaurus“ aufbauen zu lassen oder bei einem Bestehenden die Erweiterung fortzusetzen. Das Tool ist jedoch sehr einfach und hilft nur den Ablauf zu ordnen und bietet Vorschläge an.
Unschön ist auch, dass bei der Klasse das MVC-Pattern nicht eingehalten wurde.

2.4.10 BaseThesaurus

Der BaseThesaurus war die erste implementierte Klasse und ist für den Aufbau des „Basis-Thesaurus“ zuständig. Ein Basis-Thesaurus enthält lediglich Tags und Keys Informationen. Die Konzepte werden durch eine Narrower oder Broader Beziehung verbunden. Die Daten stammen ebenfalls von TagInfo.org [18]. Ein Aufbau dauert etwas weniger als 20 Minuten.

2.4.11 Vocab Package

Das Vokab-Package ermöglicht eine einfache Art und Weise um Synonyme, Ober- und Unterbegriffe für Suchworte zu erhalten. Es werden derzeit vier lizenzfreie Webservices angesprochen:

- OpenThesaurus: https://www.openthesaurus.de/about/api
- Altervista: http://thesaurus.altervista.org/service
- Wordnik: https://github.com/wordnik/wordnik-python
- Gemet: http://www.eionet.europa.eu/gemet/en/webservices/

Für jedes API gibt es eine eigene Klasse, welche die abstrakte Klasse VocabularyBase implementiert. (Besser wäre ein Interface, aber die gibt es in Python nicht.)
Diese werden in Vocabulary als Adapter instanziert: Diese Architektur ermöglicht es, schnell weitere Services anzubinden und wurde als Alternative zu einem Wikitionary-Parsing [22] umgesetzt.

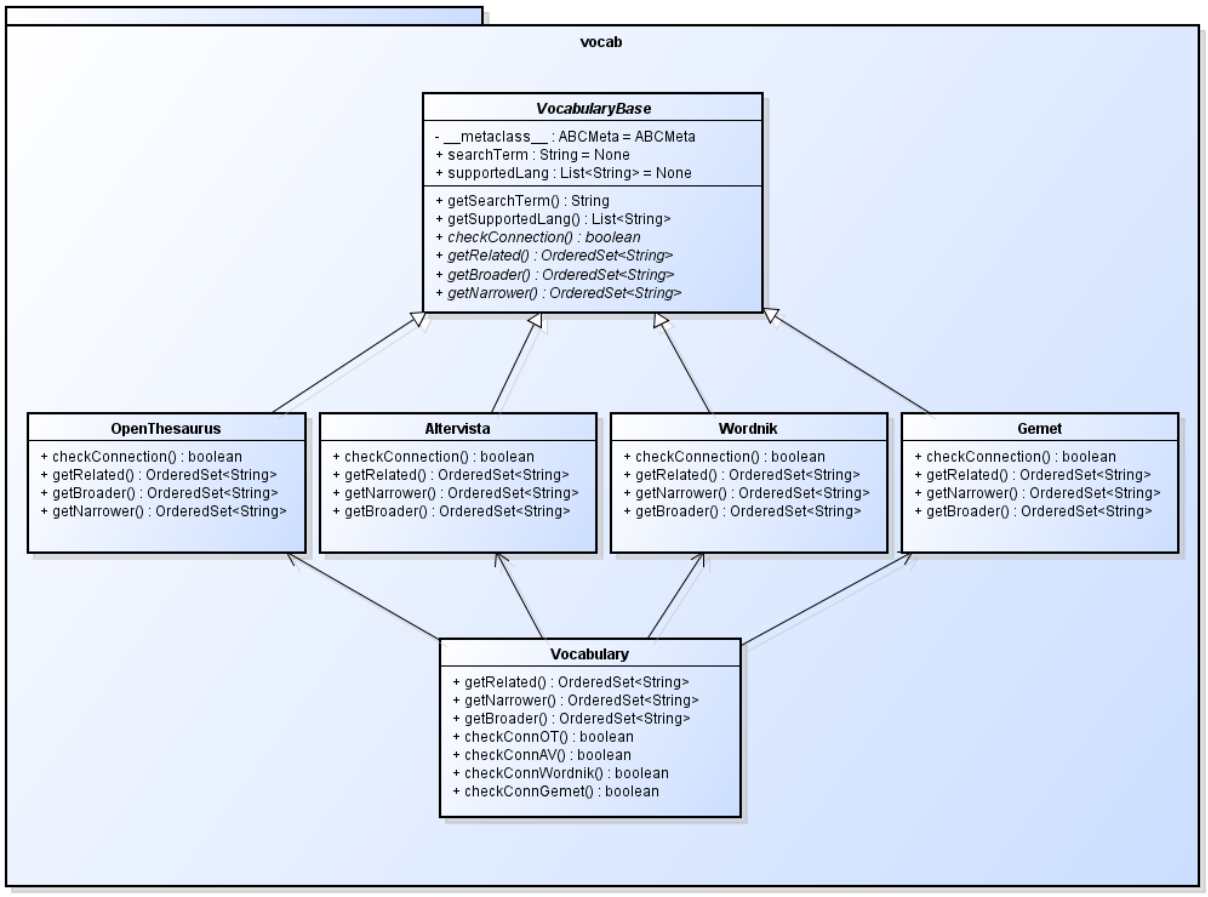


Abb II-11: Klassendiagramm der vocab-Klassen

2.5 Website-Design

2.5.1 Technischer Aspekt

Bei der Erstellung der Webseite wurde – neben dem üblichen HTML5, CSS und JavaScript – vor allem Jinja2 eingesetzt. Jinja2 wird mit Flask ausgeliefert und ist eine gute HTML-Template Sprache für Python. Mit ihr ist es möglich das serverseitig Rendering des HTML's anzupassen. Mit Jinja2 konnte schon soviel umgesetzt werden, dass das clientseitige JavaScript nur noch in einigen wenigen Fällen benutzt werden musste.

```
<!-- link list macro -->
{%- macro listlinks(linkLabelList, no_element_en, no_element_de, separator=", ") -%}
    {%- if linkLabelList|count > 0 -%}
        {%- for linkLabel in linkLabelList -%}
            {%- if linkLabel['link'] is none -%}
                {{ linkLabel['label'] }}
            {%- else -%}
                <a class="osm_links" target="_tag" href="{{ linkLabel['link'] }}">{{ linkLabel['label'] }}</a>
            {%- endif %}
            {%- if not loop.last -%}{{ separator }}&shy;{%- endif -%}
        {%- endfor %}
    {%- else -%}
        <div lang ="en">{{ no_element_en }}</div>
        <div lang ="de">{{ no_element_de }}</div>
    {%- endif -%}
{%- endmacro -%}
```

Abb II-12: Jinja2 Makro im search.html. Beim Aufruf wird eine kommagetrennte Link- bzw. Label-Liste generiert.

Die JavaScript Funktionen:

- Highlighting: Geht durch die Resultate und erkennt anhand der Such-Informationen (SearchMetas), welche Worte oder Teilworte hervorgehoben werden müssen.
- Sprachwechsel: Bei einem Wechsel der Sprache mit dem Dropdown-Menu wird eine Sprachvariable im CSS geändert. Damit muss bei einer Umschaltung nicht die ganze Seite neu geladen werden.
- Bootstrap-Typeahead: Stammt von Bootstrap 2 und fügt unterhalb eines Bootstrap-Textfelds eine „Meinten sie…?" - Liste mit Vorschlägen hinzu. Für die Vorschläge wird per AJAX der eigene Webservice angefragt.

Als Komponentenbibliothek wurde Bootstrap eingesetzt. Bootstrap Elemente wie Navigationsleiste oder Dropdown-Menus können schnell eingebaut werden und haben ein einheitliches, modernes Aussehen. Ein weiterer Vorteil von Bootstrap ist das „Mobile-First" Motto, was dazu geführt hat, dass die entwickelte Webseite responsive ist und auf vielen Smartphone-Displays gut angezeigt werden kann.

2.5.2 Visueller Aspekt

Bei der TagFinder Webseite wurde viel Wert auf das Design gelegt. Denn ein gutes Design ist das Erste, was einem Benutzer auffällt und ist deshalb zentral für die Beliebtheit der Webseite. Ein gutes Design heisst nicht, dass es nicht auch einfach sein kann, im Gegenteil. Bei der Suchmaschine ist es wichtig, dass der Benutzer die relevanten Informationen schnell extrahieren kann und bei den einzelnen Resultaten auf den ersten Blick erkennt um welchen Key oder Tag es sich handelt. Die ganze Resultateliste sollte auch „überflogen“ werden können, da stören farbige Panels und Boxen nur.

Farblich wurde mehrheitlich auf grün,schwarz und weiss gesetzt, welche stark and das OSM-Wiki erinnert: Der Nutzer soll sofort sehen, dass er auf einer Seite gelandet ist, die mit OpenStreetMap zu tun hat.
Im Folgenden werden einige Screens gezeigt, welche die Entwicklung des Designs veranschaulichen:

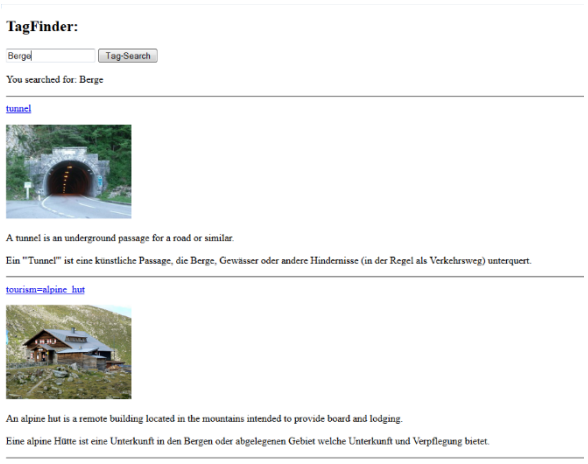


Abb II-13: Design der TagFinder Prototyp Webseite

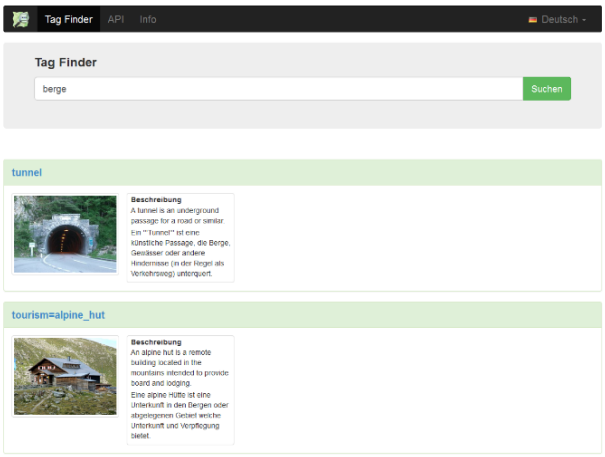


Abb II-14: Erstes Re-Design

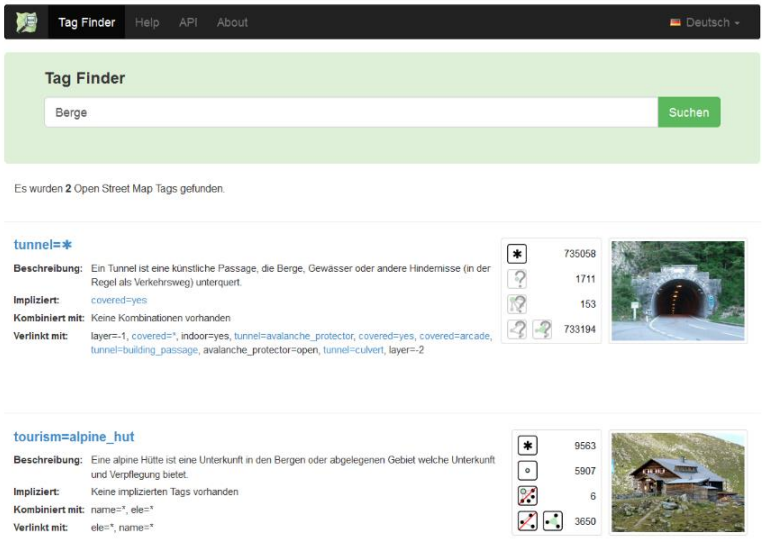


Abb II-15: Zweites Re-Design und Anzeige von mehr Informationen

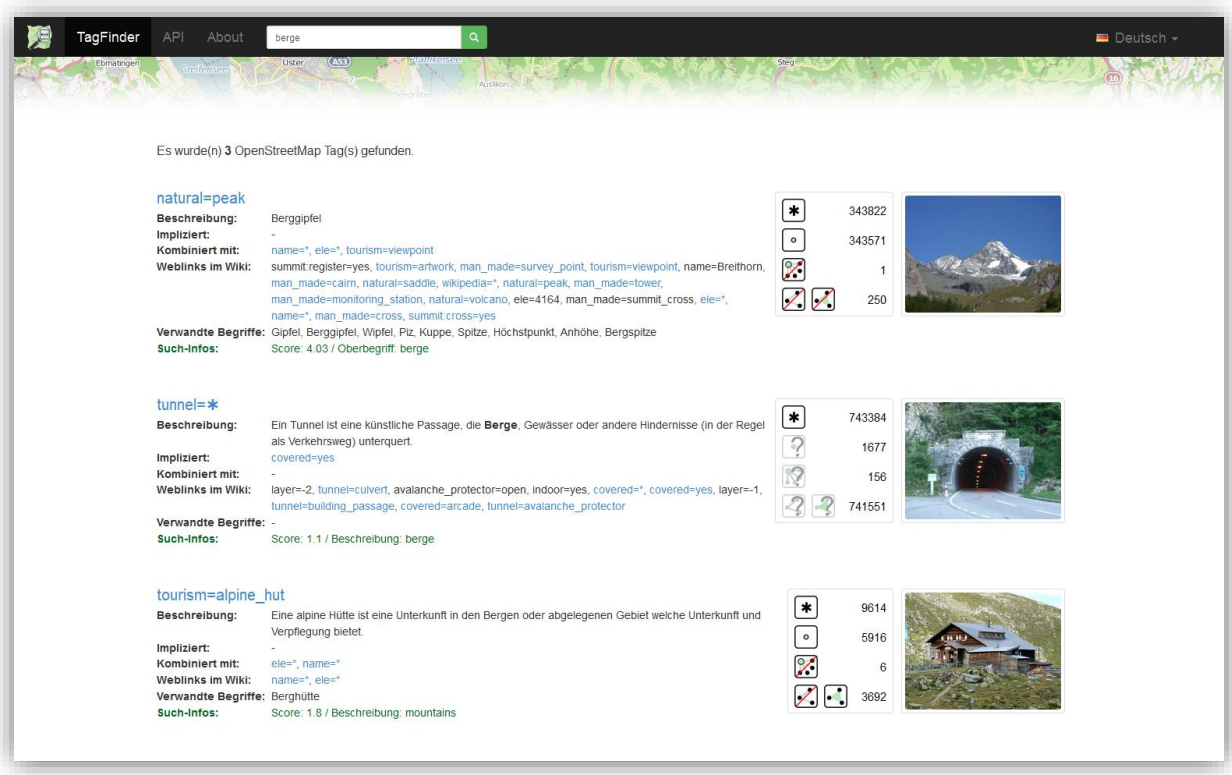


Abb II-16: Design der TagFinder Webseite am Ende des Projekts

3 Der Suchprozess

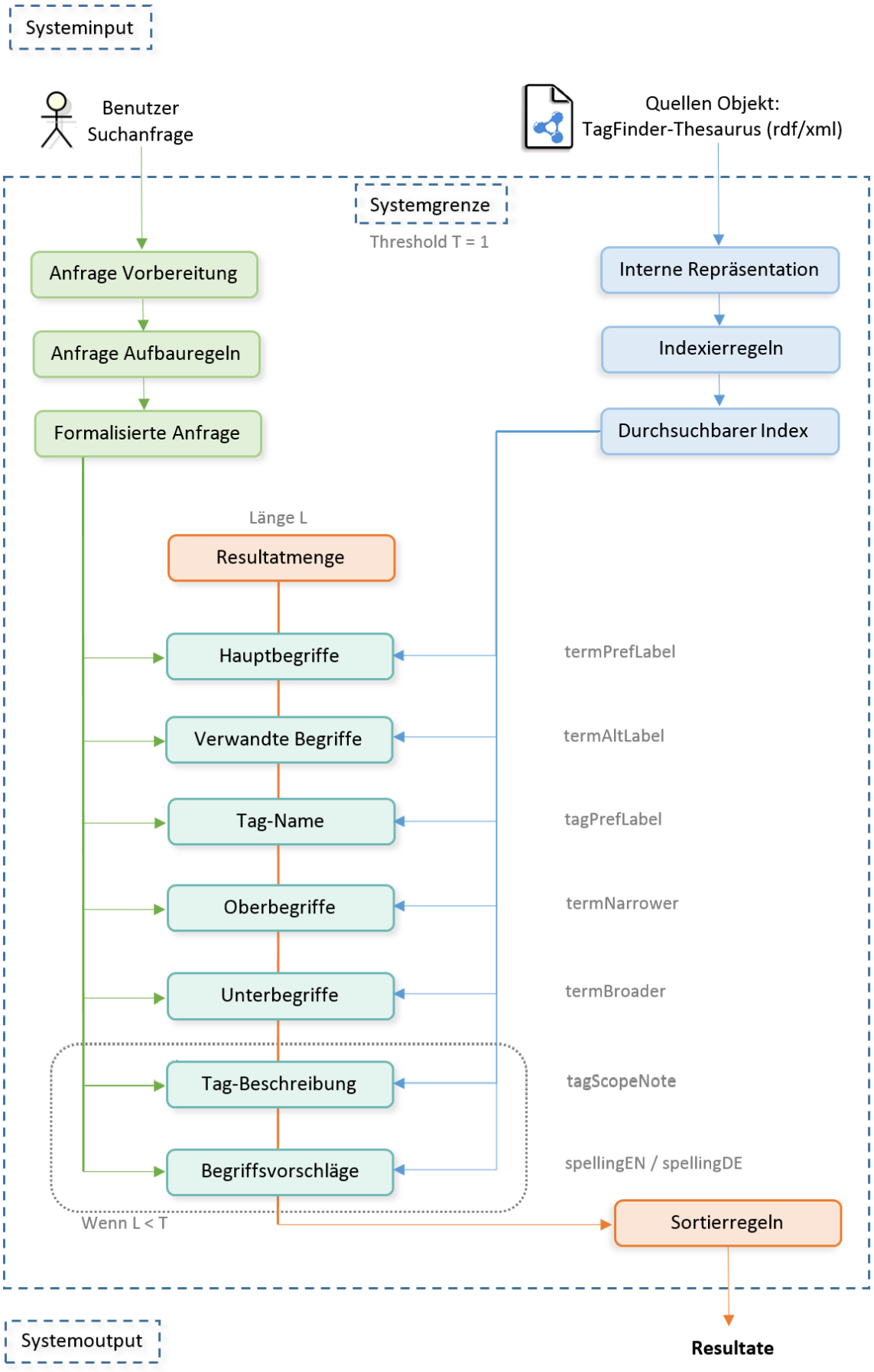


Abb II-17: Der Suchprozess von TagFinder

Abbildung II-17 zeigt den Ablauf der Suche im Detail und lehnt an einen Information Retrieval Process [23] an. Beteiligte Klassen sind der Indexer, GraphSearch und TagResults. Als Library wird wie schon erwähnt „whoosh“ verwendet.
Anhand der Abbildung wird im Folgenden der Suchprozess beschrieben:

3.1 Indexierung

Die Indexierung ist oben rechts in blauer Farbe dargestellt. Diese wird beim Starten des Servers durchgeführt oder wenn ein Thesaurus aktualisiert wurde. Das Quellen Objekt ist der TagFinder - Thesaurus im RDF/XML Format.

- Interne Repräsentation:
  Der Thesaurus wird mit der RDFGraph Klasse ins Memory geladen und dem Indexer übergeben.

- Indexierregeln:
  Der Indexer iteriert über alle Triple des Graphen und ordnet die Objekte je nach Prädikat in ein Feld des vordefinierte Schemas (Siehe Tabelle II-1) ein, dabei wird immer das Objekt (TEXT) und das Subjekt (ID) als Tuple indexiert.

Feldname	Feldtyp (whoosh)	Bedeutung
tagSubject	ID	ID eines Tag oder Key Konzepts im Graphen. Es handelt sich um den OSM-Wiki Link
tagPrefLabel	TEXT	Der Name des Tags oder Keys. Zum Beispiel „tourism=zoo“
termPrefLabel	TEXT	Die beforzugte Benennung des Tags oder Keys. Zum Beispiel „Zoo“ und „zoo“
termAltLabel	TEXT	Synonyme und verwandte Begriffe. Zum Beispiel „Tierpark“, „Zoologischer Garten“, etc.
termBroader	TEXT	Oberbegriffe des Tags oder Keys. Zum Beispiel „Tourismus“, „Sehenswürdigkeit“, „Tiere“, etc.
termNarrower	TEXT	Unterbegriffe des Tags oder Keys. Zum Beispiel „Streichelzoo“
tagScopeNote	TEXT	Die Beschreibung eines Tags oder Keys.
spellingEN	TEXT	Spezialfeld für die englischen Vorschläge.
spellingDE	TEXT	Spezialfeld für die deutschen Vorschläge.

Tabelle II-1: Die Indexerfelder. In Abb. II-17 in grauer Farbe. Ein weitere Feldtyp wäre zum Beispiel NGRAM.

Die Felder „spellingEN“ und „spellingDE“ stellen Spezialfelder für Vorschläge dar. Für sie werden alle Objekt-Literale in einzelne Worte aufgetrennt und indexiert.

- Durchsuchbarer Index:
  Hat der Indexer alle Triple abgehandelt, speichert er sie in einer Datei im Ordner ./data/indexer/ ab. Auf diese Datei kann der GraphSearch jederzeit zugreifen.

3.2 Suche

Die eigentliche Suche startet in Abbildung II-17 oben links mit der Suchanfrage eines Benutzers. Die Darstellung des folgenden Ablaufs ist in grün.

- Anfrage Vorbereitung:
  Die Suchanfrage des Benutzers trifft unverändert in GraphSearch ein. Es handelt sich um einen String mit allen möglichen Zeichen in einer Sprache. Von diesem String werden zuerst einige Spezialzeichen entfernt und dann wird er in einzelne Worte aufgetrennt. Anschliessend werden die Worte in die andere Sprache übersetzt. Am Ende stehen zwei vorbereitete Anfragen in Deutsch und Englisch bereit.
  (Es sei denn das Wort stand in Anführungszeichen, dann wird nicht übersetzt.)

- Aufbauregeln und Formalisierte Anfrage:
  Die Aufbauregelung und Formalisierung der Anfrage wird von „whoosh“ übernommen.

Nun kann der Suchprozess beginnen (türkis). Die einzelnen Felder des Indexes werden in einer bestimmten Reihenfolge mit den formalisierten Anfragen durchsucht:
Hauptbegriffe > verwandte Begriffe > Tag-Name > Oberbegriffe > Unterbegriffe

Stimmen dabei die Anfrage und der Feldinhalt überein, gibt „whoosh“ die ID (tagSubject) des gefundenen Konzepts und weitere Such-Informationen zurück. Alle gefundenen Konzepte werden in der Resultatmenge gesammelt.
Wurden jedoch diese fünf Felder erfolglos durchsucht und die Anzahl an Resultaten ist kleiner als ein Threshold, geht die Suche bei den Beschreibung- und Vorschlägefeldern weiter.

- Sortierregeln:
  Anschliessend an die Suche wird die Liste gefundener Subjekte an die Klasse TagResults übergeben, welche sich um die Sortierung kümmert. Die Sortierung behält die Reihenfolge wie die Felder durchlaufen wurde bei. Jedoch werden alle Resultate, die von demselben Feld stammen, nach totaler Anzahl Vorkommnisse in OSM absteigend sortiert.

4 Testing

4.1 Automatische Testverfahren

4.1.1 Unit-Tests

Unit-Tests dienen dem Zweck die Funktionalität der implementierten Klassen zu überprüfen und zu gewährleisten, dass Änderungen in einem Modul nicht negative Auswirkungen auf den restlichen Code hat. In Python wurde mit der Testumgebung PyUnit des PyDev Plugins gearbeitet. Beim Schreiben von Tests lag der Fokus weniger auf den UI-Klassen (d.h. dem Maintenance- und Views-Modul), sondern mehr auf den zentralen Elementen der Applikation. Leider konnten aus Zeitgründen nur die wichtigsten Klassen getestet werden. Hier gibt es definitiv Verbesserungspotential für eine grösse Testabdeckung: Total existieren im Projekt 45 Tests.
Zum Teil mussten vorderhand Mockups geschrieben werden, bzw. einen kleinen Test-Graphen, der alle möglichen Beziehungen enthält. Mit der setUp-Funktion kann dieser vor jedem Test ins Memory geladen werden (z.B. für die RDFGraph Klasse).

4.1.2 Systemtests

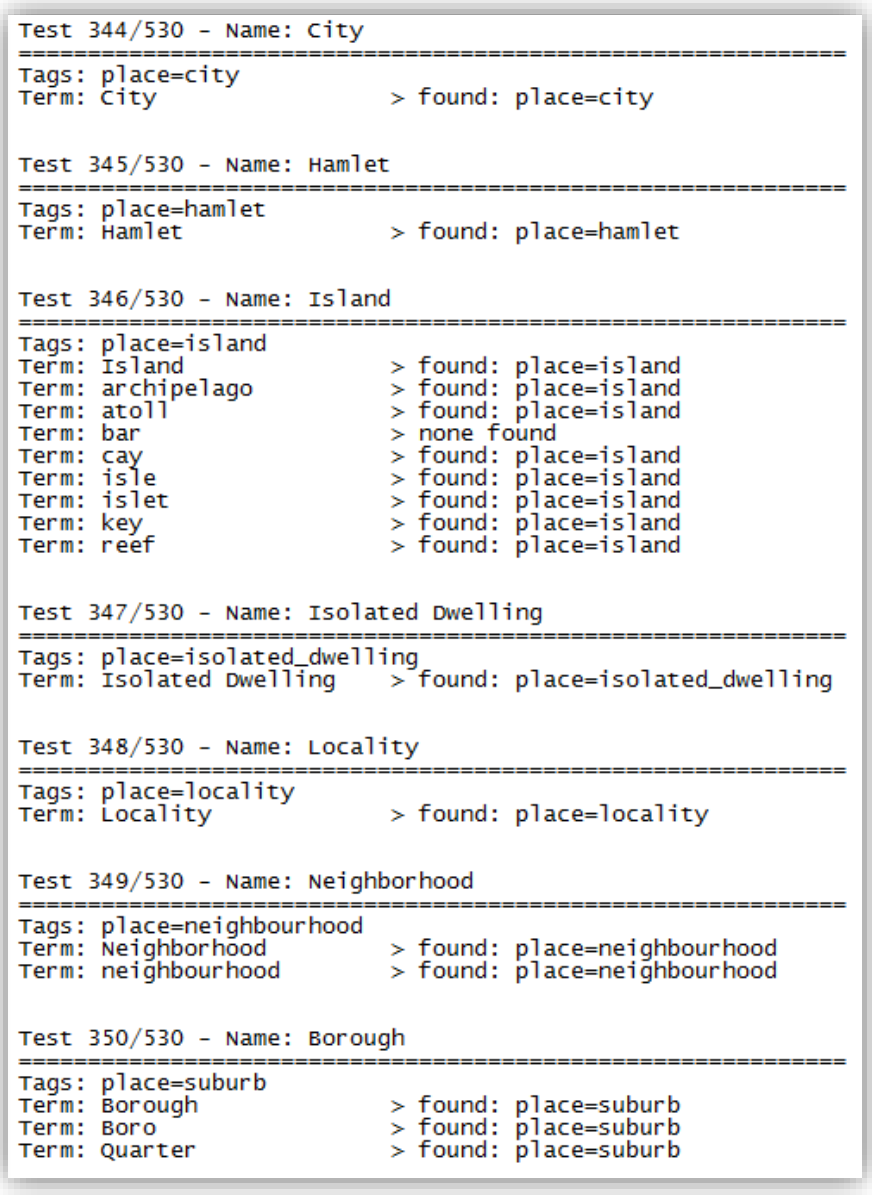
Systemtests prüfen und bewerten das Gesamtsystem und ob die gesetzten Anforderungen erfüllt werden. Im Zusammenhang von TagFinder bedeutet dies hauptsächlich die Suchfunktion zu testen: Wie gut passen die gefundenen Resultate zum Suchbegriff? Werden alle Tags gefunden?

Eigentlich war geplant dies manuell zu testen. Es wurden „Blackbox-Testdaten“ aufgenommen, welche einige Tags beinhalten und Suchbegriffe mit denen sie gefunden werden sollte. Später brachte mein Betreuer aber den Vorschlag, die Resultate zusätzlich mit den Presets des iDEditors [21] abzugleichen. Die Presets sind im JSON Format und enthalten unter anderem eine grössere Liste an wichtigen Tags und passenden Begriffen, die sich gut für Tests eigenen. Die eigenen Testdaten wurden dem JSON angehängt und anschliessend wurde eine kleine Python Testsuite geschrieben, welche die Liste der Suchbegriffe einzeln durchging, eine Anfrage an den TagFinder (per Webservice) schickte und die Resultate mit der Lösung abglich.

Die Testlogs zeigen ein gutes Abschneiden des TagFinders:

Insgesammt konnten 518 von 530 Tags gefunden werden und davon passten 1062 auf 1079 Begriffe.

Auf der folgenden Abbildung sieht man einen Auschnitt aus dem Testlog. So konnte TagFinder zum Beispiel für alle Begriffe den richtigen Tag finden ausser im Test 346: Bei „bar“ konnte `place=island` nicht gefunden werden.



```
Test 344/530 - Name: City
============================================================
Tags: place=city
Term: City                  > found: place=city


Test 345/530 - Name: Hamlet
============================================================
Tags: place=hamlet
Term: Hamlet                > found: place=hamlet


Test 346/530 - Name: Island
============================================================
Tags: place=island
Term: Island                > found: place=island
Term: archipelago           > found: place=island
Term: atoll                 > found: place=island
Term: bar                   > none found
Term: cay                   > found: place=island
Term: isle                  > found: place=island
Term: islet                 > found: place=island
Term: key                   > found: place=island
Term: reef                  > found: place=island


Test 347/530 - Name: Isolated Dwelling
============================================================
Tags: place=isolated_dwelling
Term: Isolated Dwelling     > found: place=isolated_dwelling


Test 348/530 - Name: Locality
============================================================
Tags: place=locality
Term: Locality              > found: place=locality


Test 349/530 - Name: Neighborhood
============================================================
Tags: place=neighbourhood
Term: Neighborhood          > found: place=neighbourhood
Term: neighbourhood         > found: place=neighbourhood


Test 350/530 - Name: Borough
============================================================
Tags: place=suburb
Term: Borough               > found: place=suburb
Term: Boro                  > found: place=suburb
Term: Quarter               > found: place=suburb
```

Abb II-18: Auszug aus dem Testlog der iDPreset Systemtests

Ein weiterer Vorteil dieser Art von Test war, dass die Auswirkung des Thresholds überprüft werden konnte. Das ist nützlich für das „fine-tuning“ des Suchprozesses. Es hat sich jedoch gezeigt, dass bei einer Erhöhung des Thresholds auf 2 nur 4 zusätzliche Tags zu 3 weiteren Begriffen gefunden wurden.
Daraus lässt sich ableiten, dass die minimal besseren Werte, die erhöhte Zahl unerwünschter, unrelevanten Resultate nicht rechtfertigt (Precision and Recall).

Alle Testlogs können auf der CD eingesehen werden (Siehe Kapitel IV-1).

4.1.3 Weitere Tests

- Das Testing der Webservices wurde durch ihren Einsatz auf der Webseite und im Systemtest abgehandelt.
- Die Funktionsweise von JSONP und CORS wurde mit Hilfe einer kleinen JavaScript Funktion getestet, dass Script wurde dann auf NodeJS ausgeführt.
- Die Qualität und Korrektheit der Thesaurus Syntax konnte mit verschiedenen Online-Tools geprüft werden, jedoch nur bis zu einer gewissen Grösse des Fileuploads (ca. 2MB):
  - W3C RDF Validator: http://www.w3.org/RDF/Validator/
  - qSkos von PoolParty: http://qskos.poolparty.biz/login

4.2 Manuelle Testverfahren

Die manuellen Tests werden teilweise durch die automatischen Verfahren abgedeckt. Ansonsten konnten auch hier aus Zeitgründen keine manuellen Tests durchgeführt werden.

Teil III: Projektmanagement

1 Projektorganisation

1.1 Organisationsstruktur

Folgende Personen sind am Projekt beteiligt:

Funktion	**Name**
Projektbetreuer	Prof. Stefan Keller
Projektmitarbeiter	Simon Gwerder

1.2 Externe Schnittstellen

Während des Projekts wird der Kontakt mit der OpenStreetMap Community gesucht. Da sie die Benutzer der Tag-Suchmaschine sind, sollen so viel Feedback, Anregungen und Change Requests eingeholt werden wie möglich. Die Kontaktnahme erfolgt über das OSM Community Forum und durch den Betreuer auch über die Mailing-Liste und Swiss OSM Association (SOSM).

2 Arbeitsumgebung

2.1 Hardware

An Hardware wird nur der persönliche Laptop benötigt. Die Web-Applikation wird am Ende auf Celadon Cedar-14 Heroku deployed.

2.2 Software

Folgende Software wird während der Entwicklung und für die Dokumentation verwendet:

Name	Hersteller	Version	Bereich	Beschreibung
Python 2.7	Python Software Foundation	2.7.8	Entwicklung	Python Interpreter
LiClipse	Sun / Brainwy	1.2.1	Entwicklung	Python IDE, inkl. PyDev, EGit
Sublime Text 2	Jon Skinner	2.0.2	Entwicklung	Texteditor für JavaScript, HTML
Git	Linus Torvalds	1.8.1.4	Entwicklung	Versionierungstool zur Verwaltung des Codes
Gimp	Gimp-Team	2.8.14	Dokumentation	Bildbearbeitungstool
Word	Microsoft	2013	Dokumentation	Dokumentbearbeitungstool zur Erstellung der Dokumentation
dotNetRdf	Rob Vesse / dNR Developers	1.0.7	Thesaurus Maintaining	Bearbeitung des RDF/XML, Konvertierung in .csv, .nt,…
Cytoscape	Institute of Systems Biology	3.2.0	Thesaurus Maintaining	RDF/XML Graphvisualisierung

Tabelle III-1: Verwendete Software

3 Projektmanagement

3.1 Agile Methodik: RUP

Im Rahmen dieses Projekts wird das Rational Unified Process Modell (kurz: RUP) als agile Methodik verwendet. RUP eignet sich für das Projekt aus folgenden Gründen:

- Genau bestimmtes Zeitmanagement
- Klare Meilensteine mit Liefergegenständen eingeplant
- Projektphasen passen in die vier RUP-Phasen

3.2 Zeitliche Planung und Meilensteine

In diesem Abschnitt wird die zeitliche Planung des Projektes beschrieben. Die Planung und Einteilung der Iterationen, sowie die Festlegung der einzelnen Meilensteine.
Dieses Dokument überwacht die Übereinstimmung des Zeitmanagements zwischen der Planung und der praktischen Umsetzung.

Die folgende Tabelle gibt einen Überblick der Phasen, Iterationen sowie die Resultate des Meilensteins am Ende der jeweiligen Phase:

SW	MS	Phase / Iteration	Zeitraum	Meilensteine	Resultate (nachgeführt)
1		Inception 1	15.09.2014		- Projektübersicht verschafft - Python, JS, Webdesign Tutorials - Arbeitsumgebung eingerichtet - Klarstellung der Aufgabe
2	MS1		28.09.2014	Anforderungen	
3		Elaboration 1	29.09.2014		- Technisches Umfeld - Risikomanagement - Grobprojektplan / Zeitplan - Anforderungsspezifikation
4			12.10.2014		
5		Elaboration 2	13.10.2014		- Planung abgeschlossen - Aufgabenstellung definitiv - Risiken beseitigt - Prototyp für Website, Webservice und Basisthesaurus
6	MS2		26.10.2014	Projektplan und Prototyp	
7		Construction 1	27.10.2014		- Testdaten für TagFinder Resulateabgleich - TagFinder Thesaurus Console App - Erste Thesaurus Datenbefüllung
8			09.11.2014		
9		Construction 2	10.11.2014		- TagFinder Thesaurus - Webservice API Definition - Regeln zur attributiven Heterogenität
10			23.11.2014		
11		Construction 3	24.11.2014		- Website: TagFinder, Help, API, About - Webservices UC vollständig - Entwicklung abgeschlossen - Testdokument
12	MS3		07.12.2014	Getestete Software	
13		Transition 1	08.12.2014		- Release - In Betrieb überführt - Installationsmanuals - Gesamtdokumentation
14	MS4		19.12.2014	Schlussabgabe	

Tabelle III-2: Die Phasen, Iterationen, Meilensteine und die nachträglich eingefüllten Resultate

3.2.1 Zeitplanung

Da dieses Projekt eine Einzelarbeit ist, ist es nicht nötig, eine verteilte Anwendung – wie z.B. Redmine – für die Zeitplanung aufzusetzen. Es wird eine Exceltabelle verwendet, um die Zeitplanung festzuhalten. Die einzelnen Arbeitspackete werden anfangs pro Phase definiert, jedoch spätestens im Projektplan grob ausgearbeitet. Im Laufe des Projekts werden die Arbeitspackete in spezifischere Tasks erweitert oder neue eingetragen. Die Zeiten werden nach Beendung des Arbeitspackets nachgeführt oder wenn es zu Zeitüberschreitungen kommt.

Für die Soll-Zeiten wird nach dem ECTS-Muster vorgegangen: Für Arbeit werden 8 ECTS Punkten verliehen. 1 ECTS stellt einen Aufwand von ca. 30 Stunden dar und somit soll insgesammt von 240 Stunden ausgegangen werden. Lässt man die Risiken mit einfliessen, muss mit 300 Stunden gerechnet werden. (ca. 21 Stunden pro Woche)

3.2.2 Meilensteine

Erklärung der Resultate eines Meilensteins:

- Anforderungen: Anforderungen und Aufgabenstellung definiert
- Projektplan und Prototyp: Projektplan mit Anforderungspezifikation und Prototyp der Software.
- Getestete Software: Alle Softwarekomponenten sind abgeschlossen, ein Testportokoll liegt for.
- Schlussabgabe: Abgabe aller Resultate des Projekts und Dokumentation

3.3 Besprechungen

Die Projektbeteiligten treffen sich wöchentlich zu folgenden Zeiten:

Teilnehmer:	Simon Gwerder Prof. Stefan F. Keller
Zeit:	Dienstags, 10:00 – 12:00 Uhr
Traktanden:	- Was wurde seit letztem Meeting bzw. gemäss Plan erreicht? - Offene Fragen und Hindernisse - Was ist für nächste Woche geplant?
Dokumentation:	Die Traktanden des jeweiligen Reviews oder der Besprechung werden vorzeitig vorbereitet und anschliessend den Beschlüssen nachgeführt. (Siehe Kapitel IV-8)

Liste III-1: Besprechungen

3.4 Risikomanagement

(Siehe Risikoanalyse.xls auf CD)

3.5 Codequalität

Für die Sicherstellung der Codequalität werden Unittests durchgeführt. PyDev bietet mit PyUnit ein gutes Testingframework.

Für den Codestyle wird der PyLint-Checker angewandt. Dieser überprüft, ob der Pythoncode den PEP8-Styleguides einhält [24]. (Ausgenommen der maximalen Zeilenlänge von 80.)

4 Projektmonitoring

4.1 Codestatistik

Mit dem Pythontool PyLint konnten folgende Python Codestatistiken eruiert werden:

Package	Lines of Code	Lines of Comments	Docstrings	Anzahl Module
Externalapi	591	10	71	9
Utilities	476	18	80	8
Thesaurus	1209	35	95	9
Web	630	12	23	4
root	629	15	26	3
Total	3535	90	295	33

Tabelle III-3: Python Codestatistiken von PyLint

In JavaScript wurden ungefähr 150 Zeilen Code geschrieben in 10 Funktionen.

4.2 Gitstatistik

Da der TagFinder eine Einzelarbeit ist, wird Git hauptsächlich als weiteres Backup der Entwicklung verwendet, neben der HSR-owncloud.
Die untenstehende Abbildung zeigt Daten aus der Githistorie, welche durch den GitInspector bezogen wurden. Die erste Box zeigt die Gesamtzahlen von Commits, Insertions, Deletions, und Kommentarzeilen. Laut GitInspector bestehen im Projekt nun 5471 Zeilen.
Die zweite Box zeigt die Anzahl modifizierter Zeilen ab Semesterwoche 2. Grün heisst in diesem Fall prozentualer Anteil Insertions im Vergleich zu den roten Deletions.

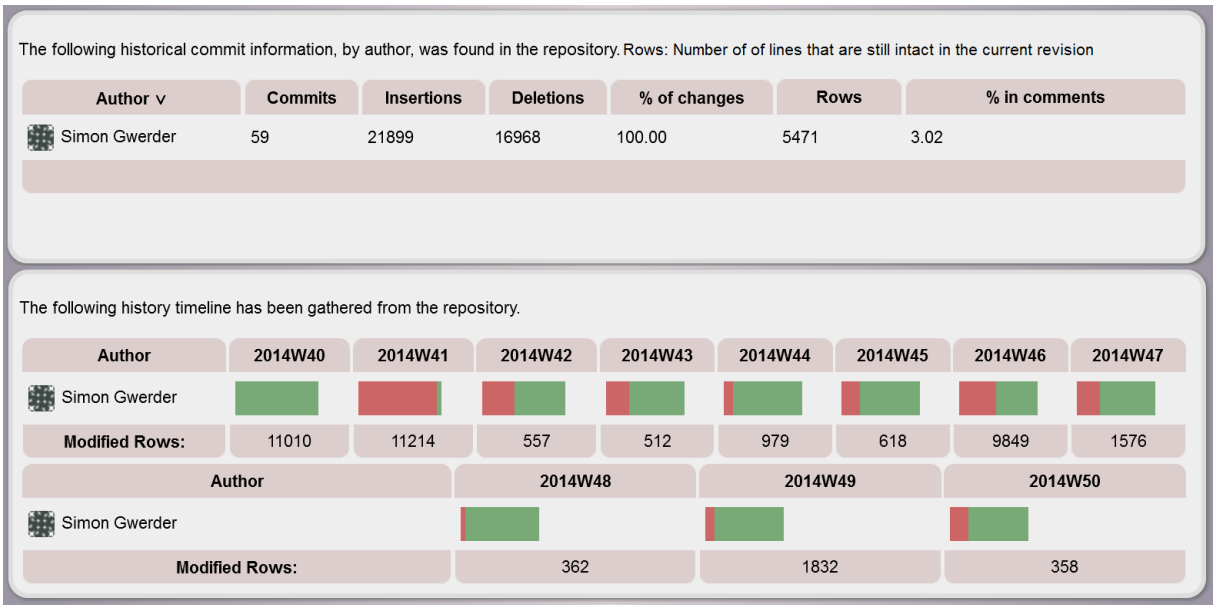


Abbildung III-1: Git-Statistiken von GitInspector

4.3 Zeiterfassung

Phase / Semesterwoche	MS bis SOLL	MS bis IST	Phase bis Forecast	Iteration	Arbeitspacket	Inhalt / Resultate		Aufwand (In h) SOLL	Aufwand (In h) IST	Delta Aufwand (in h)	fertig (in %)
Inception (I)											
SW 01				I1	Review 01: Kickoff Meeting	Beschlüsse, SA Bedingungen			2	3	1	100
				I1	Projektübersicht verschaffen	Skizze Systemkomponenten			2	2	0	100
				I1	Entwicklungs- und Projektmanagement Umgebung	Aufgesetzte Umgebung			5	4	-1	100
				I1	Technisches Umfeld: Ontology	Ontology / Thesaurus Begriff und Tools			5	7	2	100
							Total		14	16	2	
SW 02				I1	Projektdokumentstruktur	Projekt- und Qualitätsmanagement Draft			5	6	1	100
				I1	Technisches Umfeld: Evaluation bestehende Lösungen	Evaluationsentscheidung			4	4	0	100
				I1	Tutorials Python, JavaScript, CSS, Webtechnologien	Wissen / Fähigkeiten angeeignet			7	5	-2	100
				I1	Risikomanagement Machbarkeit	Simpler "Basis-Thesaurus" mit Python			4	8	4	100
	28.09.2014	28.09.2014		I1	Review 02	Beschlüsse			2	3	1	100
							Total		22	26	4	
							Total I		**36**	**42**	**6**	**100**
Elaboration (E)												
SW 03				E1	Risikoanalyse	Analyse der Risiken			3	4	0	100
				E1	Technisches Umfeld: Attributive Heterogenität	Wissen			2	3	1	100
				E1	Technisches Umfeld: Evaluation Osmantic Plugin	Evaluationsentscheidung			3	3	0	100
				E1	Projektdokumentstruktur	Teil I, II, III Draft			3	2	-1	100
				E1	Anforderungsspezifikation	Einleitung, Aktoren, Use Cases			3	3	0	100
				E1	Prototyp: Fuzzy Search	Fuzzy Search			8	9	1	100
				E1	Review 03	Beschlüsse			2	2	0	100
							Total		24	26	2	
SW 04				E1	Anforderungsspezifikation	User Stories, NFR			7	6	-1	100
				E1	Prototyp: Thesaurus Erweiterung	"Basis-Thesaurus" ausgebaut: TagInfo.org, Filter, Config			5	6	1	100
				E1	Prototyp: Thesaurus Übersetzer	Goslate Übersetzer			2	2	0	100
				E1	Prototyp: Thesaurus SpellChecker	PyEnchant Spellchecker			2	1	-1	100
				E1	Prototyp: Thesaurus Volltextsuche	N-Gramm Volltextsuche			4	2	-2	100
	12.10.2014	12.10.2014		E1	Review 04	Beschlüsse			2	3	1	100
							Total		22	20	-2	
SW 05				E2	Detailplanung	Planung			5	4	-1	100
				E2	Prototyp: Website I	Einfache HTML-Seite			5	10	5	100
				E2	Prototyp: Website Suchfunktion	Suche: Website über Flask mit Thesaurus verbunden			6	7	1	100
				E2	Aufgabenstellung definitiv	Aufgabenstellung definitv			2	1	-1	100
				E2	Review 05	Beschlüsse			2	2	0	100
							Total		20	24	4	
SW 06				E2	Prototyp: Daten anzeigen	Description, Depiction, Tag Link OSK Wiki			5	7	2	100
				E2	Prototyp: Website II	Bootstrap Einbindung, Templating, Jinja2			3	3	0	100
				E2	Prototyp: Language Changer	JS Language Chooser Dropdown EN / DE			4	5	1	100
				E2	Prototyp: TagInfo Stats	TagInfo: Total, Ways, Nodes, Relation. Sortiert die Resultate			4	5	1	100
				E2	Prototyp: Webservice	Webservice liefert diesselben Suchresultate als JSON			3	3	0	100
	26.10.2014	26.10.2014		E2	Review 06	Beschlüsse			2	2	0	100
							Total		21	25	4	
							Total E		**87**	**95**	**8**	**100**

Phase / Semesterwoche	MS bis SOLL	MS bis IST	Phase bis Forecast	Iteration	Arbeitspacket	Inhalt / Resultate		Aufwand (In h) SOLL	Aufwand (In h) IST	Delta Aufwand (in h)	fertig (in %)
Construction (C)											
SW 07				C1	Wiktionary: Mediawiki API /Crawling	Wiktionary als Begriff Thesaurus Quelle verworfen		5	8	3	*10*
				C1	Refactoring	Refactored TagFinder Thesaurus, GraphSearch		4	3	-1	100
				C1	Begriff Thesaurus Quelle	WebServices Altervista, OpenThesaurus, GEMET, Wordnik		6	7	1	100
				C1	Website Design I	Mehr Funktionalität, Copyright Footer		4	2	-2	100
				C1	Zeitmanagement präsentabel	Zeitplanung Txt > Excel		2	4	2	100
				C1	Review 07	Beschlüsse		2	2	0	100
							Total	23	26	3	
SW 08				C1	TagFinder Thesaurus Maintaining I	Console Applikation (ohne Save)		7	10	3	100
				C1	Testdaten	Testdaten pro Kategorie attributiver Heterogenität		4	5	1	100
				C1	Website Design II	Webauftritt Umgestaltung, nach Beschlüssen		4	7	3	100
				C2	Website: Node, Ways, Relations	Zusätzlich zu Total wird Node Ways und Relations angezeigt		3	4	1	100
	09.11.2014	12.11.2014		C1	Review 08: fällt aus	Beschlüsse		2	0	-2	*0*
							Total	20	26	6	
SW 09				C2	TagFinder Thesaurus Maintaining II	UC Save und Load für Console Applikation		3	4	1	100
				C2	Website: Suchmetas	Suchmeta's (was, wo, wieso gefunden pro Tag)		4	4	0	100
				C2	TagFinder Thesaurus I	Start Durchlauf Datenbefüllung per Console Applikation		5	5	0	100
				C2	TagInfo: Related, Implies, Combine	Related, Implied, Combined Tags in TagFinder und Website		6	5	-1	100
				C2	Review 09	Beschlüsse		3	3	0	100
							Total	21	21	0	
SW 10				C2	TagFinder Thesaurus III	Datenbefüllung Fortsetzung für TagFinder Thesaurus		6	6	0	100
				C2	Attributiven Heterogenität	Regeln zur attributiven Heterogenität, abgeleitet		4	6	2	100
				C2	Website Design III	Header / Footer, Navbarsearch, Highlighting, Typeahead (JS)		3	3	0	100
				C2	Suchalgorithmus und Ranking	Suchalgorithmus und Ranking, mit Blackbox-Testing		8	8	0	100
	23.11.2014	25.11.2014		C2	Review 10	Beschlüsse		2	2	0	100
							Total	23	25	2	
SW 11				C2	TagFinder Thesaurus IV	Datenbefüllung Fortsetzung für TagFinder Thesaurus		0	5	5	100
				C3	Website Design IV	Startseite, Content About		4	4	0	100
				C3	Scheduler	Scheduler für TagInfo Stats (Counts, Related, Implied, etc.)		6	8	2	100
				C3	WebService	Restliche Webservice UC implementieren		5	5	0	100
				C3	Cross Domain Requests	JSONP, CORS		6	2	-4	100
				C3	Review 11	Beschlüsse		2	2	0	100
							Total	23	26	3	
SW 12				C2	TagFinder Thesaurus V	Datenbefüllung Fortsetzung für TagFinder Thesaurus		0	5	5	100
				C3	Website Design V	API Definition auf Website und Dokumentation		4	5	1	100
				C3	OpenSearch	OpenSearch Funktionalität		2	1	-1	100
				C3	Refactoring	Refactored Code		3	3	0	100
				C3	Unit Tests, Bugfixing	Unit Tests, Bugfixing		8	5	-3	100
				C3	Systemtests, iDEditor Preset tests	Systemtests		5	6	1	100
	07.12.2014	11.12.2014		C3	Review 12	Beschlüsse		2	3	1	100
							Total	24	28	4	
							Total C	**134**	**152**	**18**	**100**

Phase / Semesterwoche	MS bis SOLL	MS bis IST	Phase bis Forecast	Iteration	Arbeitspacket	Inhalt / Resultate		Aufwand (In h) SOLL	Aufwand (In h) IST	Delta Aufwand (in h)	fertig (in %)
Transition (T)											
SW 13				T1	Dokumentation I	Teil I der Gesamtdokumentation		12	14	2	100
				T1	Management Summary	Management Summary		4	4	0	100
				T1	Abstract	Abstract		2	4	2	100
				T1	Code Stats	Code Stats mit GitInspector und PyLint		2	3	1	100
				T1	Projektmanagement, Monitoring und Reviews	Projektmanagement, Monitoring und Reviews		4	5	1	100
				T1	Review 13	Beschlüsse		2	0	-2	100
							Total	26	30	4	
SW 14				T1	Dokumentation II	Teil II der Gesamtdokumentation und überarbeitet		10	16	6	100
				T1	Usermanuals / Installationsanleitung	Usermanuals / Installationsanleitung		3	4	1	100
				T1	CD	Abgabe CD erstellt		4	3	-1	100
				T1	Poster	Poster erstellt		3	4	1	100
	19.12.2014	19.12.2014		T1	Druck und Abgabe	Geduckt und Abgegeben		3	4	1	100
							Total	23	31	8	
							Total T	**49**	**61**	**12**	**100**

Total I	36	42	6
Total E	87	95	8
Total C	134	152	18
Total T	49	61	12
Total	**306**	**350**	**44**

4.4 Soll-Ist Vergleich und Auswertung

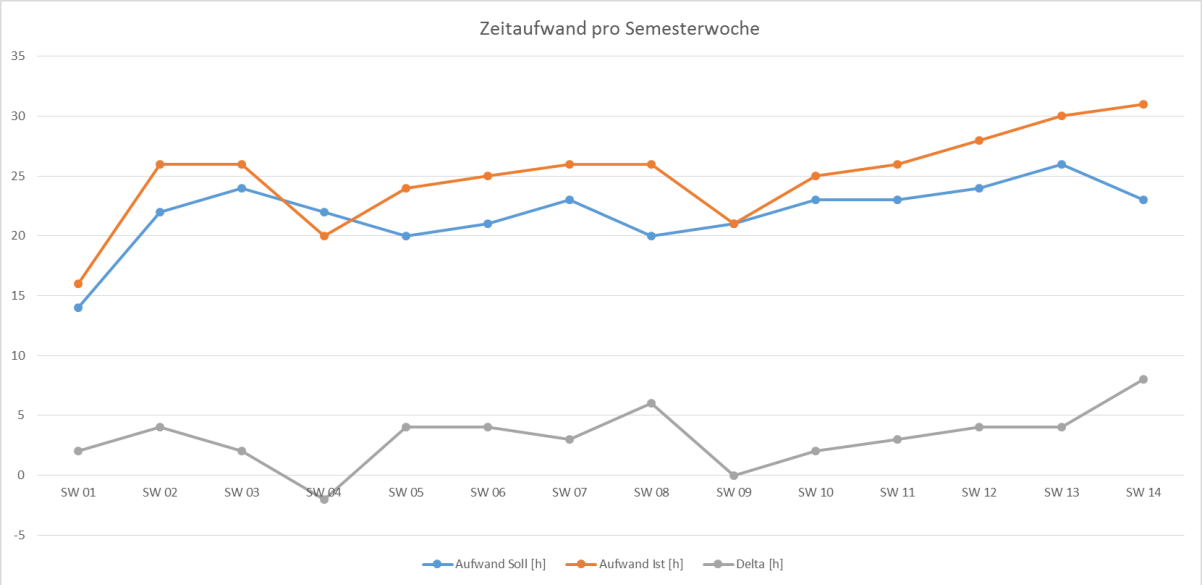


Tabelle III-3: Soll-Ist Vergleich: X-Achse: Semesterwoche, Y-Achse: Anzahl Stunden

In Tabelle III-3 sind die aufgewendeten Stunden (Ist) den geplanten Stunden (Soll) gegenübergestellt. Die graue Kurve ist das Delta zwischen beiden. Man sieht, dass fast in jeder Woche mehr Aufwand betrieben wurde als geschätzt. Ansonsten sind sich die rote und blaue Kurve jedoch sehr ähnlich. Daraus lässt sich schliessen, dass die Schätzung im Durchschnitt um 3 Stunden erhöht werden muss, ansonsten aber recht gut ist.
In Semesterwoche 4 konnten die Arbeitspackete schnell abgeschlossen werden, da die Implementation einer Volltextsuche doch weniger anspruchsvoll war. Mit „whoosh“ liess sich die Suche im Thesaurus gut umsetzen. In Semesterwoche 9 funktionierte alles wie geplant.
Was weniger gut lief war die Planung der Dokumentationsarbeit. Sie wurde massiv unterschätzt. Das zeigt sich auch am steilen Ansteig der roten Kurve in den letzten zwei Wochen. Dazu kam eine Erkrankung in der letzten Woche (SW14) mit der Folge, dass um einen späteren Abgabetermin für das finale Dokument gebeten werden musste. Zum Glück wurde das Gesuch von meinem Betreuer gutgeheissen. Dennoch musste die Dokumentation unter denkbar schlechten Umständen geschrieben werden.

Teil IV: Anhang

1 Inhalt der CD

```
/root
|
├──content.txt
|
├──Studienarbeit
|    |   Abstract.pdf
|    |   Aufgabenstellung.pdf
|    |   Installation Manual.pdf
|    |   Tag-Suchmaschine und Thesaurus für OSM_v2.0.pdf
|    |
|    └──Weitere Dokumente
|           Attributive Heterogenität.pdf
|           Meilensteine und Zeiterfassung.xlsx
|           Reviews.pdf
|           Risikoanalyse.xls
|           Some synonyms, broader and narrower terms.pdf
|           Zeitplan Übersicht.xlsx
|
├──TagFinder_v1.5
|    |   .gitignore
|    |   .project
|    |   .pydevproject
|    |   maintenance.py
|    |   maintenance.pyc
|    |   Procfile
|    |   README.md
|    |   requirements.txt
|    |   server.py
|    |   sitecustomize.py
|    |   sitecustomize.pyc
|    |   updater.py
|    |   __init__.py
|    |
|    ├──data
|    |    |   config.ini
|    |    |   tagfinder_thesaurus.rdf
|    |    |
|    |    ├──indexer
|    |    |       index_vf47yyp6zwl6yoqr.seg
|    |    |       index_WRITELOCK
|    |    |       _index_1.toc
|    |    |
|    |    ├──semnet
|    |    |       osm_semantic_network.rdf
|    |    |
|    |    └──temp
|    |            tagfinder_thesaurus_141219.rdf
```

```
|   |
|   ├──search
|   |       graphsearch.py
|   |       graphsearch.pyc
|   |       tagresults.py
|   |       tagresults.pyc
|   |       __init__.py
|   |       __init__.pyc
|   |
|   ├──semnet
|   |       osmsemanticnet.py
|   |       osmsemanticnet.pyc
|   |       __init__.py
|   |       __init__.pyc
|   |
|   ├──taginfo
|   |       taginfo.py
|   |       taginfo.pyc
|   |       taginfostats.py
|   |       taginfostats.pyc
|   |       __init__.py
|   |       __init__.pyc
|   |
|   ├──test
|   |       blackboxtests.json
|   |       idpresettest.py
|   |       preset testlog 1 - 01.12.2014.txt
|   |       preset testlog 2 - 06.12.2014.txt
|   |       preset testlog 3 - 06.12.2014.txt
|   |       presets.json
|   |       __init__.py
|   |
|   ├──thesaurus
|   |       basethesaurus.py
|   |       basethesaurus.pyc
|   |       filter.py
|   |       filter.pyc
|   |       indexer.py
|   |       indexer.pyc
|   |       mapsemnet.py
|   |       mapsemnet.pyc
|   |       rdfgraph.py
|   |       rdfgraph.pyc
|   |       relatedterm.py
|   |       relatedterm.pyc
|   |       skosserializer.py
|   |       skosserializer.pyc
|   |       updateterm.py
|   |       updateterm.pyc
```

```
│   │       updatethesaurus.py
│   │       updatethesaurus.pyc
│   │       __init__.py
│   │       __init__.pyc
│   │
│   ├───utilities
│   │       configloader.py
│   │       configloader.pyc
│   │       crython.py
│   │       crython.pyc
│   │       retry.py
│   │       retry.pyc
│   │       spellcorrect.py
│   │       spellcorrect.pyc
│   │       timeout.py
│   │       timeout.pyc
│   │       translator.py
│   │       translator.pyc
│   │       utils.py
│   │       utils.pyc
│   │       __init__.py
│   │       __init__.pyc
│   │
│   ├───vocab
│   │       altervista.py
│   │       altervista.pyc
│   │       gemet.py
│   │       gemet.pyc
│   │       openthesaurus.py
│   │       openthesaurus.pyc
│   │       vocabulary.py
│   │       vocabulary.pyc
│   │       vocabularybase.py
│   │       vocabularybase.pyc
│   │       wordnikapi.py
│   │       wordnikapi.pyc
│   │       __init__.py
│   │       __init__.pyc
│   │
│   └───web
│       │   jsonpdeco.py
│       │   jsonpdeco.pyc
│       │   views.py
│       │   views.pyc
│       │   __init__.py
│       │   __init__.pyc
│       │
│       ├───static
│       │   ├───css
```

```
|       |   |       about.css
|       |   |       apidoc.css
|       |   |       base.css
|       |   |       error.css
|       |   |       lang_de.css
|       |   |       lang_en.css
|       |   |       search.css
|       |   |
|       |   ├──ico
|       |   |       favicon.ico
|       |   |
|       |   ├──img
|       |   |       de.png
|       |   |       en.png
|       |   |       es.png
|       |   |       footer-bg.png
|       |   |       fr.png
|       |   |       header-bg.png
|       |   |       it.png
|       |   |       k200_150.png
|       |   |       k200_150_gray.png
|       |   |       kv120_120.png
|       |   |       kv200_150.png
|       |   |       kv200_150_gray.png
|       |   |       logo_hsr.png
|       |   |       logo_osm.png
|       |   |       logo_taginfo.png
|       |   |       logo_tf_128.png
|       |   |       osm_area_false.png
|       |   |       osm_area_true.png
|       |   |       osm_area_unspec.png
|       |   |       osm_count_all_v1.png
|       |   |       osm_count_all_v2.png
|       |   |       osm_node_false.png
|       |   |       osm_node_true.png
|       |   |       osm_node_unspec.png
|       |   |       osm_relation_false.png
|       |   |       osm_relation_true.png
|       |   |       osm_relation_unspec.png
|       |   |       osm_way_false.png
|       |   |       osm_way_true.png
|       |   |       osm_way_unspec.png
|       |   |
|       |   └──js
|       |           bootstrap3-typeahead.min.js
|       |           highlight.jquery.js
|       |           highlight.js
|       |           language.js
|       |           searchfade.js
```

```
|       |           spellcorrect.js
|       |           test_jsonp.js
|       |           typeahead-noselect.js
|       |
|       └──templates
|               404.html
|               405.html
|               about.html
|               apidoc.html
|               base.html
|               opensearch.xml
|               search.html
|
└──Testlogs
        blackbox 1.txt
        blackbox 2.txt
        blackbox 3.txt
        Blackbox_Testdaten.pdf
        preset testlog 1 - 01.12.2014.txt
        preset testlog 2 - 06.12.2014.txt
        preset testlog 3 - 06.12.2014.txt
```

2 Glossar und Abkürzungsverzeichnis

Begriff	Erläuterung
API	Das Application Programming Interface ist der englische Ausdruck für Schnittstelle für Anwendungsprogramme. Sie definiert den Teil des Softwaresystems, der anderer Komponenten ermöglicht sich anzubinden.
CORS	Cross-Origin Resource Sharing: Ist ein Mechanismus der Webbrowsern den Datenaustausch erlaubt, der sonst per Same-Origin-Policy untersagt wäre (Siehe JSONP, SOP)
CSS	Cascading Style Sheet ist eine Gestaltungssprache, die im Zusammenhang mit HTML verwendet wird. Ziel ist es mit ihr Darstellungsvorgaben von Webseiteninhalten zu trennen.
ECTS	European Credit Transfer System
Git	Git ist eine Software für das Versionsmanagement. Sie erlaubt die verteilte Entwicklung mit Hilfe von lokalen oder entfernten Repositories.
HSR	Hochschule für Technik Rapperswil
HTML	Hypertext Markup Language wird in der verwendet, um Webseiten-elemente zu strukturieren.
JS	Abkürzung für „JavaScript“, eine Skriptsprache, die oft in der Web-entwicklung angewandt wird.
JSON	JavaScript Object Notation ist ein Format für den Datenaustausch zwischen Anwendungen.
JSONP	JSON mit Padding: Ermöglicht die Übertragung von Daten im JSON Format über Domaingrenzen hinweg. Dieser Austausch ist bei Webbrowsern per Same-Origin-Policy untersagt (Siehe CORS, JSON, SOP).
OpenSearch	Ist eine XML basierter Spezifikation für einheitliche Suchmaschinenanfragen und –ergebnisse.
POI	Point of Interests markieren Punkte auf Karten, welche für ihre Benutzer interessant sein können. Sie befriedigen alltägliche oder reisespezifische Bedürnisse. Zum Beispiel eine Arztpraxis oder ein Kino.
REST	Representional State Transfer: Bezeichnung für ein Programmierparadigma für Webanwendungen. Zentral sind dabei die richtige Definition von Resourcen (-> URI) und der HTTP Methoden. Ziel ist die Regelung von Maschine-zu-Maschine Kommunikation.
RUP	Rational Unified Process ist ein Vorgehensmodell für die Softwareentwicklung. Zentral sind die Phasen: Inception, Elaboration, Implemention und Transition, sowie ein inkrementelles und iteratives Vorgehen.
SA	Abkürzung für Studienarbeit
SOP	Same-Origin-Policy ist ein Sicherheitskonzept im Web, dass clientseitigen Skriptsprachen untersagt, auf Objekte anderer Webseiten zuzugreifen (Siehe CORS, JSONP).
Webservice	Webservices sind Softwareanwendungen, die über ein Netzwerk Dienste anbieten. Diese Dienste werden von Computern in Anpruch genommen.

Tabelle IV-1: Glossar und Abkürzungsverzeichnis

3 Literatur- und Quellenverzeichnis

[1] OpenStreetMap. Statistics - number of users (letzter Zugriff: 07.12.2014) [http://www.openstreetmap.org/stats/data_stats.html]

[2] Wikimedia Foundation, Wikipedia. OpenStreetMap - Registered Users (letzter Zugriff: 07.12.2014) [http://upload.wikimedia.org/wikipedia/commons/c/cc/OpenStreetMap_registered_users_en.svg]

[3] OpenStreetMap. Urheberrecht und Lizenz (letzter Zugriff: 07.12.2014) [https://www.openstreetmap.org/copyright], 05.12.2014

[4] OpenStreetMap. Editing – Top three editors (letzter Zugriff: 07.12.2014) [http://wiki.openstreetmap.org/wiki/Editing], 05.12.2014

[5] Wikimedia Foundation. Wikipedia, Thesaurus (letzter Zugriff: 07.12.2014) [http://de.wikipedia.org/wiki/Thesaurus]

[6] Berners-Lee T, Hendler J, Lassila O (2001). The Semantic Web, Sci Am 284(5): 34-35

[7] Springer Verlag. (2014). Informatik Spektrum, Was bedeutet eigentlich Ontologie?, Volume 37, Issue 4, PP. 286-297,

[8] W3C Foundation, SKOS Core Guide. (letzter Zugriff: 07.12.2014) [http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102], 06.12.2014

[9] GeoNames Ontology - Geo Semantic Web. (letzter Zugriff: 07.12.2014) [http://www.geonames.org/ontology/documentation.html]

[10] linkedgeodata.org : About. (letzter Zugriff: 07.12.2014) [http://linkedgeodata.org/About]

[11] Topf, J. Semicolons in OSM Tags (letzter Zugriff: 07.12.2014) [http://blog.jochentopf.com/2013-09-23-semicolons-in-osm-tags.html]

[12] Vandecasteele A. & Devillers R. (2015). Improving volunteered geographic information quality using a tag recommender system: The case of OpenStreetMap". In: Jokar Arsanjani, J., Zipf, A., Mooney, P., Helbich, M., OpenStreetMap in GIScience: experiences, research, applications. ISBN:978-3-319-14279-1, PP. pending, Springer Press.

[13] Ballatore, A., Bertolotto, M., & Wilson, D. C. (2013). Geographic knowledge extraction and semantic similarity in OpenStreetMap. Knowledge and information systems, 37(1), 61-81.

[14] OpenStreetMap. Depricated Features (letzter Zugriff: 17.12.2014) [https://wiki.openstreetmap.org/wiki/Deprecated_features]

[15] Loidl, M., Krampe, S., Zagel, B., & Pucher, G. Aufbereitung von OpenStreetMap-Daten für GIS-Modellierungen und Analysen.

[16] OpenStreetMap. Map Features (letzer Zugriff: 17.12.2014) [http://wiki.openstreetmap.org/wiki/DE:Map_Features]

[17] OpenStreetMap. How To Map A. (letzer Zugriff: 17.12.2014) [http://wiki.openstreetmap.org/wiki/DE:How_to_map_a]

[18] Topf, J. OpenStreetMap Taginfo (letzer Zugriff: 17.12.2014) [https://taginfo.openstreetmap.org/]

[19] Ballatore, A., Bertolotto, M., & Wilson, D. C. (2013). Geographic knowledge extraction and semantic similarity in OpenStreetMap. Knowledge and Information Systems, 37(1), 61-81.

[20] Zimmermann, O. (2014). W2b, Layers und Tiers Middleware Java Enterprise Edition (JEE). Folie 19.
[21] GitHub. OpenStreetMap/iD (letzer Zugriff: 17.12.2014) [https://github.com/openstreetmap/iD/blob/master/data/presets/]
[22] Wikimedia Foundation. Wiktionary (letzter Zugriff: 20.12.2014) [http://www.wiktionary.org/]
[23] Slides: Search Engines: Information Retrieval in Practice (2014) (letzter Zugriff: 18.12.2014) [http://www.search-engines-book.com/slides/]
[24] Python Software Foundation, Python.org (2014) (letzer Zugriff: 21.12.2014) [https://www.python.org/dev/peps/pep-0008/]

4 Installation Manual

This manual covers the installation on Microsoft Windows. However installation on other operating systems should be similar or is even easier. It's assumed, that the user has basic knowledge about Python.

Content

1. Local Installation
2. Run local TagFinder server
3. Prepare Heroku and GIT
4. Deploy TagFinder on Heroku
5. Run Thesaurus maintenance
6. Run manual update of Thesaurus

4.1 Local installation

Requirements:

- Administrator access
- TagFinder projekt (e.g. from CD)
- Python 2.7.8 is installed: https://www.python.org/download/releases/2.7.8/
- PIP is installed: https://pip.pypa.io/en/latest/installing.html
- System environment variables are set to following folders (or according ones):
  - C:\Program Files (x86)\Python 2.7.8
  - C:\Program Files (x86)\Python 2.7.8\Scripts
- virtualenv: http://docs.python-guide.org/en/latest/dev/virtualenvs/ (optional)

Steps:

1) Create a project folder and copy the TagFinder projekt into it. The path to the root of the project, where you see all TagFinder files and packages, will be called <projectpath> from now on.
2) Run the cmd console with admin rights (Right-click > Run as Administrator)
3) Navigate to <projectpath>
4) Install 3rd party libraries. Enter command: `pip install -r requirements.txt`
5) TagFinder is now installed

4.2 Run local TagFinder server

Requirements:

- Local installation

Steps:

1) Check the the config file's HOST variable. HOST should be set to 0.0.0.0 or localhost. You can also set your port with variable PORT.
2) Run cmd console with admin rights
3) Navigate to <projectpath>
4) Run server with command: `python server.py`
5) Open any browser and type `localhost:5000` (default port) to see the website

4.3 Prepare Heroku and GIT

Requirements:

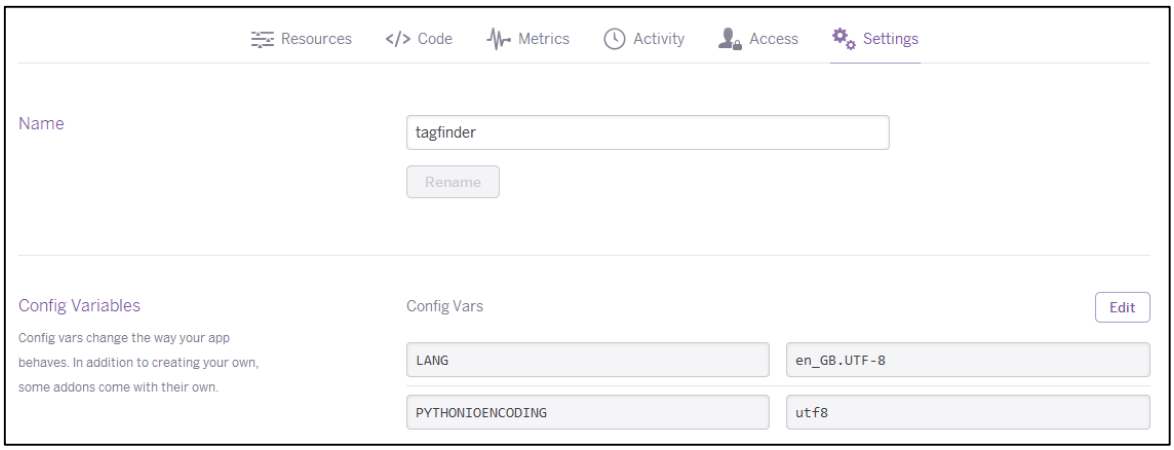
- Local installation
- Verified account on Heroku: https://www.heroku.com/
- Heroku Toolbelt is installed: https://toolbelt.heroku.com/
- GIT is installed: http://git-scm.com/downloads
- System environment variables are set to following folders (or according ones):
  - C:\Program Files (x86)\Heroku\bin
  - C:\Program Files (x86)\git\cmd

Steps:

1) Login to Heroku
2) Create a new app: Dashboard > Plus sign at the upper right > give the app a name: The name will be called <appname> from now on > choose region > Create App

   Steps 1) and 2) can also be done with console commands:
   ```
   heroku login
   heroku create -- stack cedar-14
   heroku apps:rename <appname> --app <previousname>
   ```
   (previous name will be something like fast-lowlands-9257)

3) Set config variables for the app: Personal Apps > <appname> > Settings > Reveal Config Vars > Set `LANG` to `en_GB.UTF-8` and `PYTHONIOENCODING` to `utf8`. See image below:



4) Run cmd console with admin rights
5) Navigate to <projectpath>
6) Create local GIT repository. Enter command: `git init`
7) Add the project files to the repository. Enter command: `git add -all`
   (Check with command: `git status`)
8) Commit the changes to repository. Enter command: `git commit -m "init"` ("init" is a message)
9) Login to heroku in console (if not done so previously): Enter command: `heroku login`
   You will be asked to enter your login informations and probably need to create SSH keys (on new installation). Follow the procedure.

4.4 Deploy TagFinder on Heroku

Requirements:

- Local installation
- Your app <appname> is prepared on Heroku and you created a local GIT repository in <projectpath>

Steps:

1) Run cmd console with admin rights
2) Navigate to <projectpath>
3) Add the project files to the repository. Enter command: `git add –all`
   (Check with command: `git status`)
4) Commit the changes to repository. Enter command: `git commit –m "init"` ("init" is a message)
5) Login to heroku in console (if not done so once before): Enter command: `heroku login`
   You will be asked to enter your login informations. Follow the procedure.
6) Connect repositories. Enter command: `heroku git:remote –a <appname>`
7) Upload files. Enter command: `git push heroku master`
8) Scale dyno. Enter command: `heroku ps:scale web=1`
9) Check state. Enter command: `heroku ps`. Check logs. Enter command: `heroku logs`

4.5 Run Thesaurus maintenance

Requirements:

- Local installation

Steps:

1) Run a console. It's not recommended to run the maintenance on the Windows cmd console. Python ships with "IDLE Python".
2) Navigate to <projectpath>
3) Run maintenance with command: `python maintenance.py` (or start it accordingly)

4.6 Run manual update for Thesaurus

Will update and then index the main thesaurus: <projectpath>/data/tagfinder_thesaurus.rdf
Should be done once in a while if you deployed on a free heroku dyno and therefor didn't activate the scheduler. Redeploy can be done after update finished (about 20 mins).
If you wan't to perform maintenance on the update tagfinder_thesaurus.rdf just copy it to <projectpath>/data/temp/

Requirements:

- Local installation

Steps:

1) Run cmd console with admin rights
2) Navigate to <projectpath>
3) Run maintenance with command: `python updater.py`

5 Liste der externen Libraries

In der Liste werden die alle verwendeten Python Libraries aufgeführt, inkulsive Dependencies (blau):

- Flask 0.10.1
- Flask-Bootstrap 3.2.0.2
- Flask-Cors 1.9.0
- Flask-SQLAlchemy 2.0
- Jinja2 2.7.3
- MarkupSafe 0.23
- SPARQLWrapper 1.6.4
- SQLAlchemy 0.9.8
- Werkzeug 0.9.6
- Whoosh 2.6.0
- argparse 1.2.1
- astroid 1.2.1
- chardet 2.3.0
- colorama 0.3.2
- decorator 3.4.0
- etk.docking 0.2
- futures 2.2.0
- gaphas 0.7.2
- goslate 1.3.0
- html5lib 0.999
- isodate 0.5.0
- itsdangerous 0.24
- logilab-common 0.62.1
- ordered-set 1.3
- pyparsing 2.0.2
- rdflib 4.1.2
- requests 2.4.1
- simplegeneric 0.8.1
- six 1.8.0
- termcolor 1.1.0
- wordnik 2.1.2
- xmltodict 0.9.0
- zope.component 4.2.1
- zope.event 4.0.3
- zope.interface 4.1.1

6 Webservice API

Auch auf Englisch verfügbar unter http://tagfinder.herokuapp.com/api

6.1 /api/search

(Use Case 2.1)

Pfad:	`/api/search`
Beschreibung:	Sucheanfrage an TagFinder für OpenStreetMap Tags.
Parameter:	`query` - Der Suchbegriff vom Typ STRING. (erforderlich) `lang` - Gibt die Sprache des Suchbegriffs an. Wechselt nur die Reihenfolge, in der zuerst nachgeschlagen wird. Unterstützt: 'de' und 'en'. (optional, default: 'en') `format` - Bei Wert json_pretty wird die JSON Rückgabe eingerückt. (optional) `callback` - JSONP Callback Parameter. (optional)
Bemerkung:	Wenn für den Suchbegriff keine Tags gefunden wurden, wird ein leeres ARRAY zurückgegeben. Mit Anführungszeichen um ein Wort kann die Rechtschreibkorrektur und der Übersetzer deaktiviert werden.

Resultat:

`:`	ARRAY OF HASHES	Sortierte Liste mit den Resultaten der Suchanfrage.
`subject:`	STRING	Weblink zur OpenStreetMap Wiki-Seite des Tags.
`isKey:`	BOOL	true falls es sich um einen Schlüssel handelt, false wenn es ein Schlüssel=Wert Paar ist.
`isTag:`	BOOL	true falls es ein Schlüssel=Wert Paar ist, false wenn es sich um einen Schlüssel handelt.
`prefLabel:`	STRING	Bezeichnung des Tags, im Format: Schlüssel oder Schlüssel=Wert
`depiction:`	STRING	Weblink zu einem Bild, dass den Tag auszeichnet.
`termRelated:`	ARRAY OF HASHES	
`en:`	ARRAY OF STRING	Liste mit englischen, verwandten Begriffen.
`de:`	ARRAY OF STRING	Liste mit deutschen, verwandten Begriffen.
`scopeNote:`	ARRAY OF HASHES	
`en:`	STRING	Englische Kurzbeschreibung.
`de:`	STRING	Deutsche Kurzbeschreibung.
`countAll:`	INT	Anzahl Total.
`node:`	ARRAY OF HASHES	
`count:`	INT	Anzahl Punkte.
`use:`	BOOL	Soll der Tag Punkten zugeordnet werden?
`way:`	ARRAY OF HASHES	
`count:`	INT	Anzahl Linien.
`use:`	BOOL	Soll der Tag Linien zugeordnet werden?
`area:`	ARRAY OF HASHES	
`count:`	INT	Anzahl Flächen.
`use:`	BOOL	Soll der Tag Flächen zugeordnet werden?
`relation:`	ARRAY OF HASHES	
`count:`	INT	Anzahl Relationen.
`use:`	BOOL	Soll der Tag Relationen zugeordnet werden?
`implies:`	ARRAY OF HASHES	
`label:`	STRING	Bezeichnung des implizierten Tags, im Format: Schlüssel=* oder Schlüssel=Wert.
`link:`	STRING	Weblink zur OpenStreetMap Wiki-Seite des implizierten Tags. Kann null sein.
`combines:`	ARRAY OF HASHES	
`label:`	STRING	Bezeichnung des häufig kombinierten Tags, im Format: Schlüssel=* oder Schlüssel=Wert.
`link:`	STRING	Weblink zur OpenStreetMap Wiki-Seite des häufig kombinierten Tags. Kann null sein.
`links:`	ARRAY OF HASHES	
`label:`	STRING	Bezeichnung des verlinkten Tags, im Format: Schlüssel=* oder Schlüssel=Wert.
`link:`	STRING	Weblink zur OpenStreetMap Wiki-Seite des verlinkten Tags. Kann null sein.
`searchMeta:`	ARRAY OF HASHES	
`score:`	INT	Sortierbarer Trefferscore, berechnet nach einer Okapi BM25f Variante.
`<field>:`	ARRAY OF STRINGS	Liste mit gefundenen Begriffen.<field> ist 'tagPrefLabel', 'termPrefLabel', 'termAltLabel', 'termBroader', 'termNarrower' oder 'tagScopeNote'.

Beispiel:	/api/search?query=Zoo&format=json_pretty&lang=de
UI Beispiel:	/search?query=zoo

6.2 /api/suggest

(Use Case 2.2)

Pfad:	`/api/suggest`
Beschreibung:	Gibt Vorschläge für einen Suchbegriff. Es wird eine Fehlerkorrektur verwendet.
Parameter:	`query` - Der Begriff für den Vorschläge gesucht werden soll, vom Typ STRING. (erforderlich) `lang` - Gibt die Sprache an, für welche Vorschläge gesucht werden sollen. Unterstützt: 'de' und 'en'. (optional, default: alle) `format` - Bei Wert json_pretty wird die JSON Rückgabe eingerückt. (optional) `callback` - JSONP callback parameter. (optional)
Bemerkung:	Falls keine passenden Vorschläge gefunden wurde, wird ein leeres ARRAY zurückgegeben. (Siehe auch: opensearch.xml)
Resultat:	: ARRAY OF STRING Liste mit Vorschlägen zum Suchbegriff.
Beispiel:	/api/suggest?query=tabaco&lang=en

6.3 /api/terms

(Use Case 2.3, Use Case 2.4, Use Case 2.5)

Pfad:	`/api/terms`
Beschreibung:	Erhalte Informationen zu verwandten Begriffen, Ober- und Unterbegriffen.
Parameter:	`term` - Der Begriff für den Informationen angefordert werden, vom Typ STRING. (erforderlich) `format` - Bei Wert json_pretty wird die JSON Rückgabe eingerückt. (optional) `callback` - JSONP Callback Parameter. (optional)
Bemerkung:	Die Gross- und Kleinschreibung des erforderlichen Parameters muss beachtet werden. Falls dazu keine passenden Begriffe gefunden wurden, sind die jeweiligen Listen leer.
Resultat:	: HASH termRelated: ARRAY OF HASHES en: ARRAY OF STRING Liste mit englischen, verwandten Begriffen. de: ARRAY OF STRING Liste mit deutschen, verwandten Begriffen. termBroader: ARRAY OF HASHES en: ARRAY OF STRING Liste mit englischen Oberbegriffen. de: ARRAY OF STRING Liste mit deutschen Oberbegriffen. termNarrower: ARRAY OF HASHES en: ARRAY OF STRING Liste mit englischen Unterbegriffen. de: ARRAY OF STRING Liste mit deutschen Unterbegriffen.
Beispiel:	/api/terms?term=cat&format=json_pretty
UI (kein Beispiel):	/search?query=cat

6.4 /api/tag

(Use Case 2.6)

Pfad:	`/api/tag`
Beschreibung:	Erhalte Taginformationen für einen spezifischen Schlüssel oder Schlüssel=Wert Paare.
Parameter:	`key` - Der Schlüssel des Tags vom Typ STRING. (erforderlich) `value` - Der Wert des Tags vom Typ STRING. (optional) `format` - Bei Wert json_pretty wird die JSON Rückgabe eingerückt. (optional) `callback` - JSONP Callback Parameter. (optional)
Bemerkung:	Es handelt sich hier um einen direkten Zugriff und keine Suchanfrage, d.h. die Gross- und Kleinschreibung muss beachtet werden. Falls der Tag nicht gefunden wurde, wird ein leerer HASH zurückgegeben. Die Rückgabe entspricht sonst der von `/api/search` ohne 'searchMeta'.
Resultat:	`:` HASH `subject:` STRING — Weblink zur OpenStreetMap Wiki-Seite des Tags. `isKey:` BOOL — true falls es sich um einen Schlüssel handelt, false wenn es ein Schlüssel=Wert Paar ist. `isTag:` BOOL — true falls es ein Schlüssel=Wert Paar ist, false wenn es sich um einen Schlüssel handelt. `prefLabel:` STRING — Bezeichnung des Tags, im Format: Schlüssel oder Schlüssel=Wert `depiction:` STRING — Weblink zu einem Bild, dass den Tag auszeichnet. `termRelated:` ARRAY OF HASHES `en:` ARRAY OF STRING — Liste mit englischen, verwandten Begriffen. `de:` ARRAY OF STRING — Liste mit deutschen, verwandten Begriffen. `scopeNote:` ARRAY OF HASHES `en:` STRING — Englische Kurzbeschreibung. `de:` STRING — Deutsche Kurzbeschreibung. `countAll:` INT — Anzahl Total. `node:` ARRAY OF HASHES `count:` INT — Anzahl Punkte. `use:` BOOL — Soll der Tag Punkten zugeordnet werden? `way:` ARRAY OF HASHES `count:` INT — Anzahl Linien. `use:` BOOL — Soll der Tag Linien zugeordnet werden? `area:` ARRAY OF HASHES `count:` INT — Anzahl Flächen. `use:` BOOL — Soll der Tag Flächen zugeordnet werden? `relation:` ARRAY OF HASHES `count:` INT — Anzahl Relationen. `use:` BOOL — Soll der Tag Relationen zugeordnet werden? `implies:` ARRAY OF HASHES `label:` STRING — Bezeichnung des implizierten Tags, im Format: Schlüssel=* oder Schlüssel=Wert. `link:` STRING — Weblink zur OpenStreetMap Wiki-Seite des implizierten Tags. Kann null sein. `combines:` ARRAY OF HASHES `label:` STRING — Bezeichnung des häufig kombinierten Tags, im Format: Schlüssel=* oder Schlüssel=Wert. `link:` STRING — Weblink zur OpenStreetMap Wiki-Seite des häufig kombinierten Tags. Kann null sein. `links:` ARRAY OF HASHES `label:` STRING — Bezeichnung des verlinkten Tags, im Format: Schlüssel=* oder Schlüssel=Wert. `link:` STRING — Weblink zur OpenStreetMap Wiki-Seite des verlinkten Tags. Kann null sein.
Beispiel:	/api/tag?key=shop&value=pet
UI (kein Beispiel):	/search?query=shop%3Dpet

7 Aufgabenstellung

Tag-Suchmaschine und Thesaurus für OpenStreetMap

Studienarbeit von Simon Gwerder

Abteilung Informatik, Herbstsemester 2014/15

Ausgangslage

OpenStreetMap (OSM) ist ein wiki-artiges Projekt, um gemeinsam eine Karte der gesamten Welt zu erstellen. Es ist die bekannteste Alternative zu Google Maps, die mit ihren weltweit über 1.5 Millionen Nutzern ein riesiges Potential hat. Da es ein crowdsourced Projekt ist, bei dem es keine Hierarchien gibt, kann jeder erfassen praktisch was er will. Das führt zu einem grossen Detailreichtum mit über tausend Objektarten (Tags). Das ist ein grosser Vorteil gegenüber Behördendaten und kommerziell erfassten Daten - hat aber auch Nachteile: Dasselbe Objekt wird unterschiedlich attribuiert (z.B. inhomogenes Tagging von Fuss-/Velowegen), unterschiedliche Tags können dasselbe bedeuten (Synonyme) und es fehlen Oberbegriffe (z.B. gibt es kein Begriff "Kleider-Shop", mit dem man auch Modegeschäfte, Boutiquen oder Schuhgeschäfte findet).

Aufgabenstellung

Hauptsächlich stellen sich zwei Probleme:

- Erfassung: Aus Sicht des "Mapper" wird es immer schwieriger, herauszufinden, wie ein reales Objekt attribuiert werden soll. Es gibt nur eine exakte Suche für die "Legende" und keine Suchmaschine mit "unscharfer" Suche, Sprachübersetzung und "Meinten Sie...".
- Nutzung und Analyse: Um die Daten zu professionellen Zwecken nutzen und analysieren zu können, braucht es eine definierte Qualität und eine Begriffshierarchie (z.B. "Lebensmittel-Shop" als Oberbegriff oder "Wanderweg" als Aggregation von verschiedenen Tags).

Für die erleichterte Erfassung wird ein Thesaurus (Begriffs-Netzwerk, Vokabular) aufgebaut, der aktuell gehalten werden muss. Auf dessen Basis soll eine Suchmaschine (mit Webservice/OpenSearch) für OSM-Daten realisiert werden. Ein innovativer Ansatz zur Erstellung und Nachführung des Thesaurus ist dabei, dass - nebst OSM-Daten-Statistiken (TagInfo) - Wörterbücher wie Wiktionary genutzt werden.

Zur verbesserten Nutzung und Analyse der OSM-Daten werden sie mit modernen ETL-Methoden homogenisiert und konsolidiert.

Lieferobjekte

- Thesaurus (z.B. im RDF/XML-Format) mit Werkzeugen zur Aktualisierung
- Tag-Suchmaschine (Website)
- Webservice zur Tag-Suchmaschine (REST)
- Anhand exemplarischer Beispiele soll - als Ergebnis der Nachbereitung - die attributive Heterogenität aufgezeigt werden
- Weitere nach Absprache (optional): u.a. Communityempfehlungen und verbesserte Suchfunktion in Editoren (z.B. JOSM-Plugin)

Dazu kommen die vom Studiengang geforderten Lieferobjekte.

Vorgaben/Rahmenbedingungen

- JavaScript (HTML5, REST), Python, ev. Java
- Der Student entscheidet sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer und allfälligen Projektpartnern vereinbart.
- User Interface, Source Code, Code-Kommentare und Versionsverwaltung, README (inkl. Installationsanleitung) sind in Englisch. Alles andere ist deutsch.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs bzw. der HSR.

Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind (einheitliche Nummerierung).
- Die Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen (nicht ausschliesslich Wikipedia-Links auflisten).
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Eigenständigkeitserklärung, Nutzungsrechte, ev. Poster/Video) gemäss Vorgaben des Studiengangs und Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare), inkl. je einer CD.
- Alle Dokumente und Quellen der erstellten Software auf den 3 CDs (+1 Ex. Abteilung); CD's sauber angeschrieben.

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Studienarbeit des Studiengangs Informatik mit besonderem Gewicht auf moderne Softwareentwicklung wie folgt:

- Projektorganisation (Gewichtung ca. 1/5)
- Bericht, Gliederung, Sprache (Gewichtung ca. 1/5)
- Inhalt inkl. Code (Gewichtung ca. 2/5)
- Gesamteindruck inkl. Kommunikation mit Industriepartner (Gewichtung ca. 1/5). Ein wichtiger Bestandteil der Arbeit ist, dass eine lauffähige, getestete Software abgeliefert wird (inkl. getesteter Installationsanleitung).

Beteiligte

Student

Simon Gwerder, sgwerder@hsr.ch

Projektpartner

OpenStreetMap Community (Open Data)

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller, IFS-HSR (sfkeller@hsr.ch)

8 Reviews

8.1 Kickoff – 16.09.2014

Offene Fragen

- Initiale Datenerfassung als Basis der Ontologie erfolgt über bestehende Datenbanken wie EOSMDBOne, Semantic Network oder irgendwie „crawlen“?
- Was ist der Scope der Suchmaschine, bzgl. Sprachen, länderübergreifend, Sprachübersetzung von Tags.
- Soll die Verknüpfung von z.B. Synonymen zu Begriffen über z.B. DBPedia erfolgen?
- PyLucene, Scrum / Redmine, Build Process, Externe Libraries.
- SA/BA_Tipps.pdf?
- Wikiseite für Arbeitsspezifikation editieren
- TagInfo

8.2 Review 01 – 23.09.2014

Was wurde erreicht

- Begriffe und Umfeld recherchiert:
  - OpenStreetMap: Funktionen verwenden. Webseite / iD / JOSM. Gebäude mit Strassennummer hinzugefügt.
  - Tags:
    - TagInfo: Statistiken der Tags, werden aus der OSM „Planet“ DB generiert. Hat sehr gute API.
    - TagFinder: Offline / Bug, NameFinder Offline
    - Nominatim / Special Phrases -> Übersetzungen Addr. Tag + Name Tag
- Ontologien / Semantic Web / Linked Data / GeoNames / …
- OSM Semantic Web: Leider nicht durchgehend SKOS Format, bzw. nicht korrekt. 2 Jahre alt. Basiert auf Groovy / Java => OSM Wiki Crawler. WebInterface Offline.
- Verschiedene Editoren für SKOS:
  - Protégé mit SKOSED Plugin: langsam, buggy => Problem?
  - dotRdfEditor: Wenig Funktionalität, bzw. ist angepasster XML Editor, aber kann Konvertierungen vornehmen in verschiedene Ontologieformate (rdf, nt, …)
  - ThManager: Installiert nicht
  - TopBraidComposer / OntoStudio / PoolParty / Synaptica /… kommerziel und teuer.
- Python: LiClipse IDE, Tutorials. Erste Webservices mit web.py. PyUnit / EGit
- Python 2.7.8 oder 3.4.1?
- Versionsmanagement: git.hsr.ch
- Dokumentstruktur
- Systemkomponentendiagramm

Was ist geplant

- Dokumentation Teil 1 – Technischer Bericht: Technisches Umfeld, Vision, Architektur und Technologien, Theorie.

Beschlüsse

- 2 Artikel lesen und Analyse: Loidl, A. Ballatore
- Nachfolge von TagFinder
- OSMCheatSheet.pdf
- Sektion Verwandte Begriffe im OSM Wiki, heisst „Related Terms“
- Thesaurus: Wortliste: BS (Benutze Synonym), OB (Oberbegriff), UB (Unterbegriff)
- OSMonto Paper
- Python 2.7.8 und Flask statt web.py

8.3 Review 02 – 30.09.2014

Was wurde erreicht

- Literatur:
  - Geographic Knowledge Extraction and Semantic Similarity in OpenStreetMap
    Andrea Ballatore, Michela Bertolotto, David C. Wilson (2012)
  - A Survey of Volunteered Open Geo-Knowledge Bases in the Semantic Web
    Andrea Ballatore, David C. Wilson, Michela Bertolotto (2012)
  - Aufbereitung von OSM-Daten für GIS-Modellierung und Analysen
    M. Loidl, S. Krampe, B. Zagel und G. Pucher (2014)
  - Informatik Spektrum – Ontologien.
    Springer-Verlag. Band 37, Heft 4 (2014)

- Evaluation und technisches Umfeld:
  Thesaurus, RDF, OWL, SKOS. SKOS eignet sich doch sehr gut als formale Sprache. Zusätzlich können weitere formale Sprachen herangezogen werden für spezielle Relationen z.B. depiction aus FOAF für Bilder-Links oder relatedMatch für Mappings zu anderen Ontologien. Es kann ohne weiteres in einige gängige Formate serialisiert werden: rdf/xml (primär), n3, nt, turtle, trig, trix und nquads

- Tutorials für JavaScript mit Nodejs als Runtime durchgearbeitet, inkl. WebServices.

- Weitere Tests mit Python: Libraries wie flask, requests und json für WebServices, rdfLib für das Parsen und Serialisieren von RDF-Ontologien.

- Mit Python wurde eine erste Basis-Ontologie generiert. Dazu wurden Anfragen an TagInfo gesendet und die Antwortdaten weiterverarbeitet. In der Basis-Ontologie wurden nur Tags von http://taginfo.osm.ch berücksichtigt. Ausserdem sind weitere Einschränkungen beinhaltet. Um aufgeführt zu werden brauchen die Keys und Tags zwingend eine OSM-Wiki Seite und müssen mindestens 10 mal in der Karte verwendet werden. Die Keys brauchen mindestens 10 verschiedene Values und einige wenige Keys wie name, addr:city, addr:street und openGeoDB werden ohne Values aufgeführt.
  Zu den Tags und Keys werden die englischen und deutschen Labels, die Beschreibung, eine Broader / Narrower Beziehung und ein Bildlink eingetragen.
  Momentan beinhaltet die Ontologie 228 Keys, 780 Tags und 6238 Triples. (Siehe Anhang)

  Ich denke auf dieser Basis-Ontologie lässt sich aufbauen und ich konnte meine grössten Risiken und Unklarheiten beseitigen.

- Projekt- und Qualitätsmanagement Draft

- Versionsmanagement mit GIT auf http://git.hsr.ch/git/OSMTagFinder

Hindernisse und offene Fragen

- Was gewünscht ist für die Anforderungsspezifikation erscheint mir zu diesem Thema noch recht vage. V.a. bezüglich Use Cases. (OSM-Mapper --> OSM-Tag suchen).
- Scope von Semesterarbeit allgemein besprechen: Zeitlicher knapper Rahmen und inhaltlich einheitliches Thema.
- Was soll der Scope der Ontologie sein (Sprachen, Mindestanzahl Keys und Tags und weitere Auswahlkriterien)? Einbringen von Daten aus WordNet und LinkedGeoData Tags.
- Python: 3rd Party Libraries korrekt installieren im Hinblick auf Deployment.
- Projektnamen

Was ist geplant

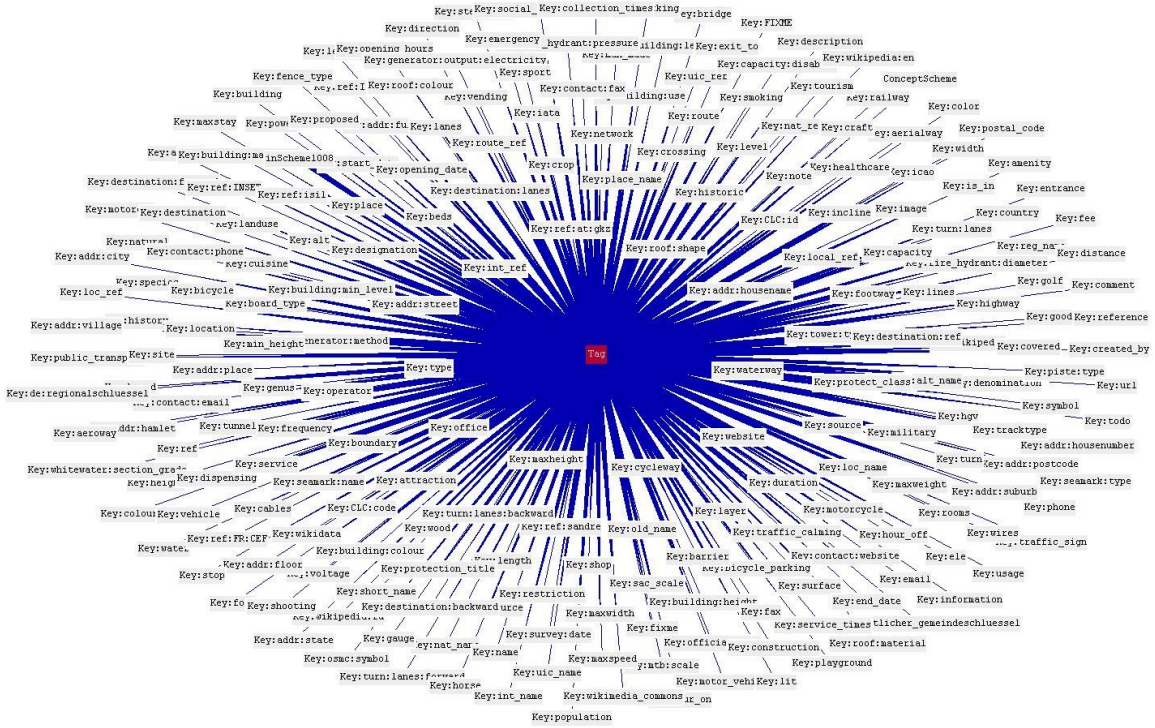
- Risikoanalyse
- Vision
- Benutzercharakteristiken, Personas und Context Scenarios
- Projektmanagement

Beschlüsse

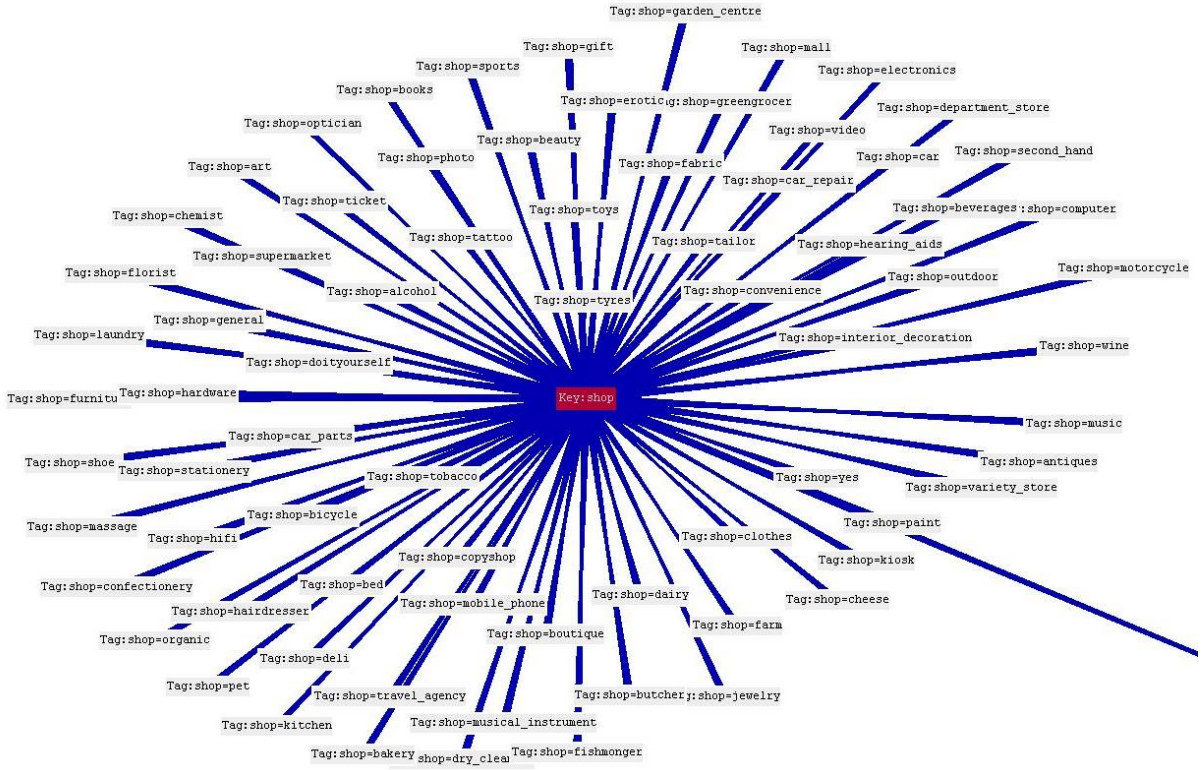
- Google Schoolar -> 1. Teil. Wissenschaftliche Referenzen / Zitate unbedingt.
- Key nicht immer Oberbegriff von Tag. Z.B. bridge=yes, building=yes
- Do-roam von Osmonto
- Empfehlug: Auswertung, Auflistung als Resultat von Ontology Detail, wo braucht es mehr.
- Liste Analyse der attributiven Heterogenität, Siehe Loidl
- TagInfo.org als Tag Quelle. Sprachen Deutsch und Englisch.

Anhang

Graphische Darstellung eines kleineren Teils der Keys aus der Basis-Ontologie:



Alle Values von Key „shop“:



8.4 Review 03 – 07.10.2014

Was wurde erreicht

- Literatur:
  - OSMonto – An Ontology of OpenStreetMap Tags
  - Ontology-based Route Planning for OpenStreetMap

- OSMantic : Plugin für JOSM (läuft nicht auf neuster JOSM Version)

- Logo für TagFinder erstellt:

- TagFinder Thesaurus ausgeweitet auf TagInfo.org (alle Tags) und Sprachen Englisch und Deutsch. Die Untergrenze der Taganzahl liegt jetzt bei 5.
  Der Filter musste auch um einiges erweitert werden.
  Der Algorithmus wurde so verbessert, dass die initiale Datenverarbeitung noch 4 Minuten dauert. `Keys: 740, Tags: 1759`

- PyEnchant Library ist für die „fuzzy“ Suche praktisch. Ein deutsches, englisches und französisches Wörterbuch ist enthalten. Es ist zudem möglich, eigene Wörterlisten anzuhängen. In unserem Fall wäre eine Begriff Wörterliste der Tags denkbar.
  Alternativen sind GNU Aspell oder Whoosh
  Beispiel: Englische Suche nach "tabaco" liefert:
  `['Tabasco', 'tobacco', 'Tacoma', 'tailback', 'Tabatha', 'tabor']`

- Dokumentstruktur und teils Inhalt für "Teil I – Technischer Bericht", "Teil II – SW-Projektdokumentation" und "Teil III – Projektmanagement" gemacht.

- Anforderungsspezifikation Kapitel Einleitung, Aktoren und Use Cases.

Hindernisse und offene Fragen

- Soll Swiss OpenStreetMap Association (SOSM) erwähnt werden in "Externe Schnittstellen" (Involvierte Externe ...)?
- Einrichtung eines Dokument Shares oder jeweils per Email zuschicken?
- Kann "Kapitel I-1.1 Problemstellung" als Vision gelten und braucht es das Unterkapitel Vision nochmals in der Anforderungsspezifikation?
- Braucht es Kapitel "Überblick" für alle 4 Hauptteile?
- Python: 3rd Party Libraries korrekt installieren im Hinblick auf Deployment?

Was ist geplant

- Anforderungsspezifikation vervollständigen (Use Cases, etc.) und Änderungswünsche aus Besprechung umsetzen.
- Anfangen mit Analyse und Design / Prototyp

Beschlüsse

- Externe Schnittstelle: OpenStreetMap Community (Open Data)
- Sprachen Deutsch und Englisch
- Document Shares: owncloud.hsr.ch
- Kapitel Überblick bei erstem Hauptteil
- Deployment: Liste mit Pip (requirements.txt). Sonst setup.py
- Tagging-Schemen mit drei Dimensionen: Zeitlich, Geografisch und Konzeptionell
- LinkedOpenData
- Nächstes Meeting 14.Oktober.14, 11:00

8.5 Review 04 – 14.10.2014

Was wurde erreicht

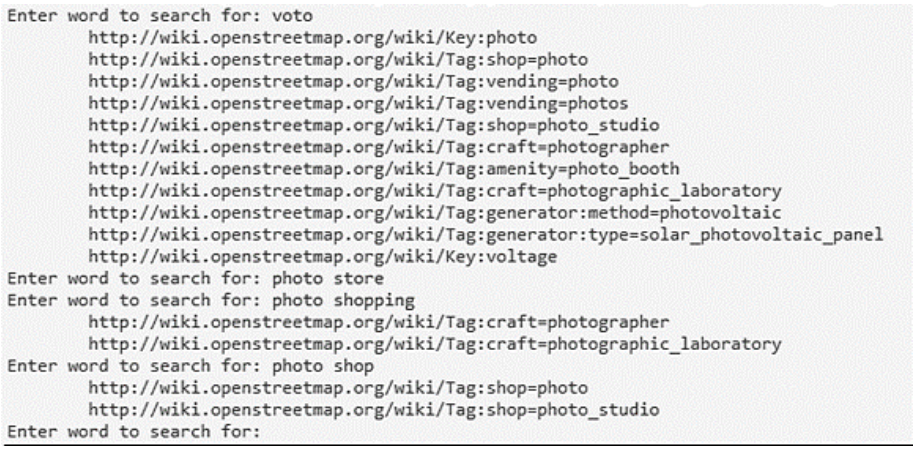
- Literatur:
  Semantic Integration of Authoritative and Volunteered Geographic Information (VGI) using Ontologies (Memorial University of Newfoundland)
  Improving volunteered geographic information quality using a tag recommender system (A. Vandecasteele – Chapter 1)
- N-Gramm Volltextsuche im „Basis-Thesaurus“. Greift auch auf einen Übersetzer und den SpellChecker zurück. Funktioniert sehr gut soweit und schnell. Als Resultate werden Taginformationen erhalten: prefLabel, altLabel, narrower, broader, image, description). Es wäre möglich Such-Meta’s zurückzugeben, d.h. wo im Tag und welcher Teilbegriff zum Hit geführt hat.
  Ein Problem ist noch, dass eine Suche mit „aussagelosen“ Begriffe viele Resultate liefern können, wegen SpellCorrection. So gibt eine Suche mit sop viele verschiedene Resultate und dann erst eine Liste an Shops.

Hindernisse und offene Fragen

- Allgemein verbesserte Tagsuche in den Deliverables definieren oder einfach als positiven Nebeneffekt erwähnen?
- Mein Domainmodel, also klassische Domainklassen, stecken grösstenteils in 3rd Party Libraries, die ich verwende.

Was ist geplant

- Webseite, Webserver mit Flask. Einfaches Suchfeld. Eine Suche gibt Resultate aus Thesaurus.
- Detailplanung



Beschlüsse

- Was gibt es für Kategorien und Ursachen von Attributiver Heterogenität.
- Domainmodell: Thesaurus und RDF-Graph Ausschnitt
- Anführungszeichen => Ohne SpellCorrector / Translator suchen
- Editoren haben verschiedene Auswahllisten / Presets => Testfälle
- Zeitveränderung z.B. bei fireplace=yes
- Aufgabenstellung in den Anhang
- Titel: Tag-Suchmaschine und Thesaurus für OSM
- SPARQL – Michael Hedinger. Seminar mit Testbeispielen.

8.6 Review 05 – 21.10.2014

Was wurde erreicht

Prototyp der Tag-Suchmaschine. Beinhaltet einfache HTML-Seite mit Suchfeld. Bei einer Suche werden die passenden Tags (und allenfalls Keys) untereinander angezeigt. Sortiert wird nach einem Ranking, wie gut der Suchbegriff auf ein Tag passt, bzg. Textmatch.
Für die gefundenen Tags werden folgende Informationen angezeigt:

- Tag- oder Key-Label mit Link auf OSM-Wikiseite
- Bild falls vorhanden, sonst ein Symbolbild
- Beschreibung in Englisch und Deutsch, falls vorhanden.

Für die Webseite wird Flask-Mikroframework verwendet.

Hindernisse und offene Fragen

- Würde eine Sortierung nach Häufigkeit wie bei Tag-Info überhaupt Sinn machen?
- Wie weit soll die Suche gehen (Siehe nächste Seite)?

Was ist geplant

- Prototyp ausbauen: Design, CSS, JavaScript, ev. „bootstrap-flask“, Language dropdown
- Suchresultate verbessern, Such-Meta (wieso wurde das Resultat gefunden) anzeigen.

Beschlüsse

- Keys nicht in der Liste anzeigen als Oberbegriff
- Sortierreihenfolge: Ranking!
- CoffeeScript als JavaScript alternative
- Anzahl Treffer anzeigen
- STS, WWF Artikel „Gute Noten für Schweizer Zoos“: 51 Zoos. In OSM tourism=zoo ist da offener was ein Zoo ist bzw. mehrfach Tagging pro Zoo-Gebäude oder Eingang => Mehr Zoos in OSM als tatsächlich (wie in Artikel). Manchmal Tagging als Punkt, manchmal als Fläche.
- leisure=golf_course ist verwandt bzw. Unterbegriff von sport=golf
- Function Facet für Ranking, Frequency
- Information Retrieval Process (Hufeisen)
- AND Query Model, nicht OR.

8.7 Review 06 – 28.10.2014

Was wurde erreicht

- Webseite (bsp: host/search?q=post):
  - Nun mit Bootstrap (flask-bootstrap python library) umgesetzt (inklusive CSS, JS).
  - Template für „Menu-Ribbon“ mit allen Pages die umzusetzen sind.
  - TagFinder Startpage zeigt bei Suche die Resultate in einzelnen Panels an. Grün für Tags, Blau für Keys.
  - Die Sprache lässt sich wechseln ohne Seitenrefresh.
  - Gefundene Tags beinhaltet nun auch die totale Anzahl Elemente wie oft sie in OSM vorkommen, Anzeige der Unterteilung in Nodes, Ways und Relations wäre zudem möglich (geplant).
  - Die Resultate werden nun nach dieser Anzahl sortiert.

- WebService (bsp: host/api/search?q=post):
  - Liefert dieselben Resultate wie die Suche über die Webseite in JSON.

Hindernisse und offene Fragen

- Anfragen an TagInfo für die „Counts“ machen die Suche bei vielen Resultaten sehr langsam.
- MediaWiki API ist kompliziert / umständlich, vielleicht besser Wiktionary selber crawlen.
- NGRAM bei “Description” gibt schlechte Resultate.

Was ist geplant

- Wiktionary API Zugriff mit Python evaluieren
  MediaWiki => Pywikibot, mwclient, python-mwapi etc.
- Suchresultate verbessern, Such-Meta (wieso wurde das Resultat gefunden) anzeigen, Anzahl Treffer

Beschlüsse

- Language-Switch Dropdown
- Navigationsleiste: TagFinder / API / About (auch auf Deutsch)
- Drehende Pfeil bei längeren Wartezeiten
- Weniger Abstand zwischen Suchresultaten, keine Rahmen
- Total Count nicht als Bootstrap Label
- Lizenzangabe wie TagInfo, Copyright, OSM Contributors (ObdL) mit Links, zudem TagInfo, HSR und Aktualität der Daten daneben angeben.
- API testen: Webseite Darstellen mit WebService statt direkt
- JSONP und CORS, Cross-Site Origin zulassen.
- Config File für Startup
- Zeitplanung!

8.8 Review 07 – 04.11.2014

Was wurde erreicht

- Wiktionary / Mediawiki als Thesaurus Quelle für Erstellung und Maintaining des TagFinder Thesaurus verworfen.

- Alternative Umgesetzt: 4 Webservices die Begriffe aufgeführt haben, wie Synonyme, Ober- und Unterbegriffe. Altervista (de/en), GEMET* (de/en), OpenThesaurus (de) und Wordnik (en). Alle 4 Webservices implementieren ein Interface (Python ABCMeta), dass um weitere Thesauri erweitert werden kann, falls je gewünscht.

- GEMET RDF Graph ladbar in meine Klasse: Geplant: Mapping von TagFinder zu GEMET statt Wiktionary. Beinhaltet dann auch tatsächliche Konzepte für Ober- / Unterbegriffe, Themen, Gruppen und Obergruppen, was mit Wiktionary nicht der Fall unbedingt gewesen wäre.

- Zeitmanagement Txt > Excel

*GEneral Multilingual Environmental Thesaurus

Hindernisse und offene Fragen

- Hindernis: Mediawiki API und Wiktionary
- Nächste Reviews, Kontaktmöglichkeit?

Was ist geplant

- Thesaurus Maintaining Console Applikation
- Testdaten pro Kategorie attributive Heterogenität für Suchmaschine. Vergleich OSM Suche, TagInfo, TagFinder jetzt.

Beschlüsse

- Gründe in Dokumentation angeben, wieso Wiktionary verworfen wurde.
- railway=station ⇔ building=train_station (Node vs. Fläche)
- Bus Stop ist zeitliche Veränderung
- Nach Sektionen sortieren, wobei Oberbegriffe > Unterbegriffe
- tourism=zoo, zoo=*, zoo=petting_zoo
- Reloading / Startup auf Server: Per Scheduler, Angabe wieviel editierbare Tags in der Consolenapplikation
- Sonst Änderungen manuell Online bringen.

8.9 Review 08 – 11.11.2014

(Meeting hat nicht stattgefunden und das Review wurde am nächsten Termin besprochen)

Was wurde erreicht

- Attributive Heterogenität - Exemplarische Beispiele
- Mapping TagFinder Thesaurus zu OSM Semantic Web von A. Ballatore und damit quasi auch zu WordNet.
- Thesaurus Maintaining Console Applikation teilweise fertig zusammen mit GEMET RDF Graph.
  Anfragen erfolgen auf allen 4 Thesauri (Webservice) aufgrund eines englischen und deutschen Begriffs, der angegeben werden muss. Aus den Resultaten kann ausgewählt werden, welche Synonyme, Ober- und Unterbegriffe übernommen werden sollen.
- Im TagFinder Thesaurus Graph wird nun zusätzlich zu den Anzahl Nodes, Ways und Relations auch abgespeichert, ob Node, Way und Relation für Tag auch verwendet werden soll (TagInfo liefert die Daten, diese sind z.T. widersprüchlich / restriktiver gegenüber denen von der Wikiseite)
- Die Relations von der OSM Wikiseite, d.h. „Implies“ (z.B. access=yes bei highway), „Useful Combinations“ (z.B. firepit bei picnic) und „Links“ (alle sonstigen verlinkten Tags auf einer Tag-Wikiseite) werden nun ebenfalls im TagFinder Thesaurus abgebildet.
- Webseite Redesign:
  - Layout nach Beschlüssen; ohne Panelrahmen etc.
  - Nodes, Ways + Areas und Relations + Bild zur Verwendung dargestellt.
  - OSM Relations Implies, Combines und Links werden als Liste in Tabelle dargestellt. Diese werden of OSM Wikiseite verlinkt, falls vorhanden.

Hindernisse und offene Fragen

- Was wäre eine gute Darstellung der gefunden Tags auf der Webseite? V.a. die „Linklisten“, wenn diese lang wird undTextfluss. (Siehe Bild nächste Seite)
- Den Basis Graphen zu erstellen mit den Neuerungen dauert jetzt bis zu 25min.
  (Von

Was ist geplant

- Zwar bin ich bei der Consolenapplikation noch nicht so weit wie gedacht, dafür wären die Tasks „Implies, Combines, Links“ und Darstellung auf Webseite zusammen mit Anzahl „Nodes, Ways, Relation“ schon gemacht, obwohl für nächste Woche geplant.

=> Thesaurus Maintaining Console Applikation Logik und Ablauf fertig
(d.h. ohne Load UC).

- Rückgabe und Darstellung der Suchmetas auf Webseite

- Durchlauf starten für Thesaurus Maintaining (Datenbefüllung)

8.10 Review 09 – 18.11.2014

Was wurde erreicht

- Webseite Design: Flow der Links um die beiden Boxen (Stats, Bild), Searchbar Fade, Fixed Navbar
- Heroku Demo Webseite
- PyEnchant durch Whoosh abgelöst: Spelling correction basiert jetzt nicht mehr auf Wörterbüchern (was aber auch möglich wäre) sondern auf existierende Begriffen aus Keys / Tags.
- Thesaurus Maintaining Console:
  - Load, Save und weitere Commands
  - Durchläufe startbar => Anfragen erfolgen auf allen 4 Thesauri (Webservice) aufgrund eines englischen und deutschen Begriffs, der angegeben werden muss. Aus den Resultaten kann ausgewählt werden, welche Synonyme, Ober- und Unterbegriffe übernommen werden sollen. Eingabe als Listenindex oder eigene Begriffe möglich.
  - Übersetzung des Tags / Keys, Wiki-Link im Copy-Clipboard als Hilfe

Hindernisse

- Website Designing „Hacks“
- Demo auf Heroku:
  - PyEnchant benutzt die Enchant C Library, was sich auf Heroku nicht einfach installieren lässt => PyEnchant spelling correction ersetzt durch Whoosh.
  - Gratis Dynos von Heroku hat nur ein Read-Only-Filesystem: Problem für Scheduler und Thesaurus Maintaining.
  - Python Encoding auf Heroku
- Python Encoding allgemein im Zusammenhang mit Umlauten und Eszett. (Terminalencoding, Sourcefileencoding, Umgebungs-variablen, DefaultEncoding 2.7) => chardet library

Offene Fragen

- Website Design: Was wäre ein gutes Design der Websitelinks, was, wenn es keine Links hat? Related, Narrower, Broader Begriffe oder nur deren Tags darstellen?
- Trotzdem auf Heroku deployen?
- Regeln und Skripte <=> „Attributive Heterogenität - Exemplarische Beispiele“

Was ist geplant

- Thesaurus Maintaining Console fertig (Finalizing)
- Testdaten
- Rückgabe und Darstellung der Suchmetas auf Website
- Durchlauf Thesaurus Maintaining (Datenbefüllung)

Beschlüsse (vom 11.11.14 und 18.11.14)

- Rating OSM completeness weogeo
- Namespaces von Tags, Kommas, Doppelpunkt, Pipe, etc. => Topf Blog
- „Links im Wiki“ => „Weblinks im Wiki“ umbennen
- Copyright und Version der Webseite bei dessen Start, soll bei Anzeige von Resultaten verschwinden
- OpenStreetMap statt Open Street Map
- Preffered Term in Console trennen von Suche nach Begriffen (Vorschläge)
- Clipoard(buffer) statt Kopierbuffer
- Escape für Commands in Console: Besser Backslash statt Bindestrich
- In der Console die Verfügbarkeit / Connection anzeigen der WebServices
- Wenn es auf der Webseite keine Links hat: Bindestrich statt Text

8.11 Review 10 – 25.11.2014

Was wurde erreicht

- Thesaurus Maintaining Console: Für ungefähr 200 / 2574 Tags wurden related terms, narrower und broader erfasst.
- Die Suche behandelt jetzt auch Haubtbegriffe und Verwandte Begriffe.
- Suchmetas werden berechnet.
- Website Design:
  - Copyrights und Logos von OSM, TagInfo und HSR, es fehlt jedoch noch Angabe zum Datum der Datenerhebung.
  - Darstellung der Suchmetas: *Score* ist ein sortierbarer Wert, der die Relevanz zum Suchbegriff angibt. Whoosh berechnet die Score mit angepasster Okapi BM25F Ranking Funktion. Als nächsten werden die Trefferfelder angezeigt, d.h *Tagname / Hauptbegriff / Verwandter Begriff / Beschreibung* und dann noch das (Teil-)Suchwort. Problematisch ist jedoch, dass nicht nach Score sortiert werden kann. Zoo Beispiel.
  - Generell Anpassung des Suchfelds und Design
- Testdaten

Hindernisse

- Datenerfassung ist nicht so einfach.

Offene Fragen

- „Attributive Heterogenität - Exemplarische Beispiele“
- Darstellung der Suchmetas
- Scheduler

Was ist geplant

- Durchlauf Thesaurus Maintaining (Datenbefüllung)
- Webservice / JSONP
- Webseite Design, About, ev. API schon anfangen (ähnlich TagInfo.org)

Beschlüsse

- Name konsequent „TagFinder“ oder „OSM TagFinder“ (Kein Leerzeichen, CamelCase)
- Statt „Suchmeta“ => Suchinfos / Search infos
- N-Gramm nicht verwenden, auch nicht wenn es sonst nichts findet!
- Daten vom…
- Highlighting auf der Webseite

8.12 Review 11 – 02.12.2014

Was wurde erreicht

- Thesaurus Maintaining Console: Für ungefähr 400 / 2574 Tags wurden related terms, narrower und broader erfasst.

- WebServices, CORS und JSONP

- Information Retrieval Diagramm

- Website Design:
  - Highlighting des Suchbegriffs in den Resultaten.
  - „Meinten sie…?“ ähnliche Box bei der Sucheingabe (per AJAX call auf WebService).
  - Kleinere Unschönheiten beseitigt und Startpage Design verbessert.
  - Ein wenig für Mobile Devices verbessert.

Hindernisse

- Bei der Datenerfassung werde ich nicht alle Tags durcharbeiten können bis zur Abgabe.
- „Zeitfresser“ HTML und JavaScript

Was ist geplant

- Weiteres Thesaurus Maintaining (Datenbefüllung)
- Suchalgorithmus und Ranking verbessern, Narrower und Broader einbeziehen
- About und API
- Testing, ev. mit iD-Editor Schemas
- Scheduler
- Code-Freeze

Beschlüsse (vom 11.11.14 und 18.11.14)

- Abstand mit Levenstein Algorithmus
- Placeholder in Textfeld wenn noch keine Eingabe erfolgt ist:
  „Suche Tags“ bzw. „Search Tags“
- Chrome Entwicklertools für Mobile
- Threshold: Recall and Precision
- Anstand zwischen Resultaten noch mehr verkleinern

8.13 Review 12 – 09.12.2014

Was wurde erreicht

- Code Freeze
- Thesaurus Maintenance: Für ungefähr 800 / 2574 Tags wurden related terms, narrower und broader erfasst.
- OpenSearch
- Thesaurus Maintenance: Finalizing
- Suche nach „broader“ und „narrower“ terms.
- Entsprechende Sortierung der Ergebnisse. Die Sektionen (= wo gefunden) sind fest in ihrer Reihenfolge: Tag, Hauptbegriff, Verwandte Begriffe, Oberbegriffe, Unterbegriffe, Beschreibung. Innerhalb der Sektionen wird mit der Anzahl Vorkommen in OSM sortiert.
- Scheduler & Updating: Scheduler mit „crython“. Crontab als Trigger-Pattern in Config.ini. Updating holt die neusten Daten von TagInfo und stellt Vergleich im Vorhanden Graph an. => Neue Tags anlegen, bestehende Tags updaten um Statistiken, Beschreibung, Bild und Links. Depricated Tags bleiben bestehen, aber Statistik „zurückgesetzt“.
- BlackBox-Tests und Presets-Tests von idEditor (Siehe Testlog Files) und Analyse: Nachteil bei Mehrzahlen: „Hochschulen“ / „Wälder“
- Suchalgorithmus: Translations / Narrower / Broader.
  Threshold Problematik (anhand Tests): Threshold=1 findet wenig für “woodland”, aber “xmas” ist ok. Threshold=2 findet auch „woodland“ gut, aber zu viel für „xmas“.
- Webservices: /api/tag/ und /api/terms/.
  API Page (Anfang)

Hindernisse

- Zeit => Community-Emfehlungen?
- Anzahl Printouts?

Was ist geplant

- API und About Page
- Auswertung und Dokumentation

Beschlüsse

- Verwandte Begriffe nach unten, vor Such-Info.
- Grafik von Vandersteele verwenden. Tagging-Schema Bezug auf „yes“ nehmen.
- OpenSearch
- Poster: Problem -> Lösung. Stichworte