Universität Passau
Fakultät für Informatik und Mathematik
Lehrstuhl für Mathematik
mit Schwerpunkt Digitale Bildverabeitung
Prof. Dr. Tomas Sauer

Qualitätssicherung von Kartendaten anhand von OpenStreetMap

Bachelorarbeit

12. August 2013

Verfasser:	Sebastian Brunner
Matrikelnummer:	61402
Studiengang:	Bachelor Informatik
Betreuer:	Prof. Dr. Tomas Sauer

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Passau, 12. August 2013

Zusammenfassung

Diese Bachelorarbeit befasst sich mit dem Thema Qualitätssicherung von Kartendaten und benutzt dazu die freie Weltkarte *OpenStreetMap*. Es wird eine Web-Applikation bereitgestellt, die es den Teilnehmern von OpenStreetMap leichter machen soll, von ihnen erstellte Strukturen zu überwachen und Änderungen, die andere Teilnehmer daran vornehmen, nachvollziehen zu können.
In dieser Arbeit werden zuerst grundlegende Dinge besprochen, um dem Leser einen Einstieg in das Thema zu ermöglichen, ohne dabei große Vorkenntnisse vorauszusetzen. Dabei wird zu Beginn die Repräsentation von geographischen Objekten in OpenStreetMap erläutert, d.h. es wird die Frage geklärt, wie zum Beispiel eine Straße so erfasst wird, dass ein Programm diese Daten als Straße interpretieren kann. Weiter werden verschiedene Datei-Formate erklärt, die solche Informationen beinhalten und welchem Zweck unterschiedliche Formate dienen. Da diese Daten bereitgestellt werden müssen, folgt ein Abschnitt über die Overpass API, einen speziellen Dienst, der darauf spezialisiert ist, OpenStreetMap-Daten auszuliefern. Zudem wird erläutert, wie die Interaktion mit dieser API abläuft - inklusive der Vorstellung einer Anfragesprache. Weiter wird darauf eingegangen, welche Möglichkeiten grundsätzlich bestehen, um in OpenStreetMap nach bestimmten Elementen zu suchen. Dies wird am Beispiel des bekanntesten OpenStreetMap-Editors JOSM aufgezeigt. Den Abschluss des Einführungs-Teils bildet je ein Abschnitt über die verwendete Web-Technologie JavaServer Pages und RSS.
Anschließend erfolgt im Praxis-Teil eine kurze Vorstellung des implementierten Programms. Zwei weitere große Abschnitte werden durch die Erläuterung und Bewertung des Algorithmus abgedeckt. Hierbei wird nicht nur auf den Ablauf des Algorithmus eingegangen, sondern auch auf Probleme, den Programmfluss, den Ressourcenverbrauch und auf andere mögliche Herangehensweisen. Abschließend werden bereits vorhandene Programme vorgestellt, die eine ähnliche Intention verfolgen, wie das Programm dieser Bachelorarbeit. Zudem werden Erweiterungsmöglichkeiten genannt, die sinnvoll sein könnten bzw. bereits implementiert wurden, aber für diese Arbeit nicht von Relevanz waren.
Im letzten Kapitel wird Bilanz gezogen: Es wird gezeigt, ob das Programm bei der OpenStreetMap-Community positiv oder negativ aufgenommen wurde.

Kapitel 1

Das OpenStreetMap Projekt

Möglicherweise benutzen sie OpenSource Software wie LibreOffice/OpenOffice auf ihrem Rechner. Dann kennen sie den Gedanken der hinter OpenSource Projekten steckt: Software mit liberalen Lizenzen, die jeder modifizieren und mitgestalten darf. Ähnlich zu OpenOffice ist das OpenStreetMap Projekt. Für den normalen Benutzer stellt OpenStreetMap (OSM)[1] eine Karte der ganzen Welt dar, genauso wie Google oder Yahoo Maps. Eigentlich ist OSM aber eine Datenbank, die geographische Informationen über die Erde enthält. Auf diese Datenbank hat jeder Interessierte Zugriff und kann zum Beispiel Anfragen stellen, wie z.B. wie viele Kilometer Autobahn es in Deutschland gibt. So kann jeder Mensch die Daten uneingeschränkt nutzen und seiner Kreativität bei deren Einsatz freien Lauf lassen [14]. Damit erreicht OSM nicht nur die Darstellung der Welt auf einer Karte, sondern bietet seinen Teilnehmern ein Grundlage für eine Vielzahl von Anwendungen, wie Routenplanern oder extra Karten für den öffentlichen Nahverkehr.
Das OSM Projekt wurde 2004 von Steve Coast gegründet. In seinen Anfängen lag der Fokus von OSM auf dem Vereinigten Königreich [11]. Mit der Zeit wuchs das Projekt und konnte durch seine Offenheit gegenüber neuen Ideen und durch die einfache Benutzung immer mehr Menschen begeistern. Mittlerweile wird die ganze Welt erfasst und große Teile davon sind auf einem aktuelleren Stand als bei Google Maps. Nach Angaben der Statistikseite des OSM-Wikis [21] sind mehr als 1,1 Millionen Benutzer bei OSM registriert.
Doch wie können so viele Menschen an einem einzigen Projekt arbeiten? Das Aufgabengebiet bei OSM ist riesig: Um eine Weltkarte im Computer anzeigen zu können, braucht man geographische Rohdaten, zum Beispiel den Verlauf einer Küstenlinie, eines Flusses oder einer Straße. Diese Aufgabe wird von den Mappern übernommen. Die Mapper sammeln mit Hilfe von GPS-Geräten oder Luftbildern Daten, bearbeiten diese und stellen sie in der OSM-Datenbank zu Verfügung.

[1] Homepage: `http://www.openstreetmap.org/`

Ein anderes Aufgabengebiet ist die Anwendungsprogrammierung. Wie eingangs erwähnt, stehen die geographischen Rohdaten, mit denen gearbeitet werden kann, jedem Menschen frei zur Verfügung, so dass Programmierer nach eigenem Belieben Applikationen für das OSM Projekt oder für sich selbst schreiben können. Die wichtigsten Anwendungen für OSM sind wohl die Renderer, d.h. Programme, die aus den Rohdaten eine gebräuchliche Karte erzeugen.
Diese Bachelorarbeit befasst sich mit dem OpenStreetMap Projekt, erklärt die Verwaltung der Informationen und stellt eine Anwendung bereit, die der Qualitätssicherung von OSM-Daten dienen soll: OSMarelmon, der OpenStreetMap Relation Monitor, der dafür gedacht ist, dass Mapper Relationen besser überwachen können. Auf das Thema Relationen und OSM Datenstrukturen im Allgemeinen, wird in Kapitel 3.1 noch eingegangen werden. Nach einer Analyse des Programms und der verwendeten Techniken, werden noch Ähnlichkeiten zu bestehender Software, mögliche andere Herangehensweisen und interessante Erweiterungsmöglichkeiten genannt.

Kapitel 2

Motivation

Das Programm OSMarelmon soll helfen, die Qualität der Daten in OSM zu sichern. Wie bereits erwähnt, arbeiten mittlerweile mehr als eine Million Menschen an OpenStreetMap mit. Auch wenn viele wahrscheinlich nicht ständig die OSM-Karte verbessern, anpassen oder ergänzen, sind die Änderungen, die täglich an den OSM-Daten vorgenommen werden, riesig. Für einen Menschen ist es nicht möglich, die Übersicht über diese Änderungen zu behalten und auch Computerprogramme stoßen bei so großen Datenmengen, wie sie bei OSM vorherrschen, schnell an ihre Grenzen. Nun sei ein klassisches Beispiel genannt, das auch im weiteren Verlauf dieser Arbeit benutzt werden wird, um diese Problematik zu veranschaulichen:
Betrachten wir die Buslinien einer Stadt, zum Beispiel Passau, da diese dort sehr genau erfasst sind. Durch die rege Beteiligung an OSM ist es sehr wahrscheinlich, dass in Passau weitere Mapper ihre Arbeit verrichten und möglicherweise Straßenverläufe korrigieren, Ampeln eintragen oder andere Arbeiten erledigen. Dabei kann es passieren, dass diese Mapper Teile der Buslinie ändern: Sei es eine Verschiebung eines einzelnen GPS-Punktes, einer ganzen Straße oder gar die Löschung derselben. Auch eine unabgesprochene Mitarbeit an den Buslinien kann vorkommen. Nun ist der erste Mapper, der die Buslinien mühevoll erfasst hat, wahrscheinlich sehr daran interessiert, dass sein Werk vollständig und korrekt erhalten bleibt und er über Änderungen dieser Buslinien gerne informiert werden würde. Aufgrund der Datenmenge von OSM und die Verwaltung der Änderungen hat der Mapper alleine keine Chance, Änderungen an seinen Buslinien mitzubekommen, ohne jeden Tag auf der Karte alles manuell abzugleichen.
An dieser Stelle setzt das Programm OSMarelmon an: Es bietet dem OSM-Mapper an, seine Buslinien zu überwachen und ihn über Änderungen per RSS Feed zu benachrichtigen. Somit kann sich der Mapper einen Überblick verschaffen, wer zu welchem Zeitpunkt etwas an seinen Buslinien verändert hat und gegebenenfalls eingreifen.

Kapitel 3

Einführung in die verwendeten Technologien

Dieses Kapitel soll einen Überblick und eine kurze Einführung in die Technologien geben, die im Programm zu dieser Bachelorarbeit verwendet wurden. Als erstes werden die OSM-spezifischen Datenstrukturen näher erläutert. Daraufhin wird in einem extra Abschnitt auf die Änderungsdateien von OSM eingegangen. Ein weiterer Abschnitt befasst sich mit der Overpass API, einem Dienst, der OSM-Daten bereitstellt. Danach wird auf eine speziell für OSM entwickelte Suchfunktion eingegangen. Den Abschluss bildet eine kurze Einführung in die JavaServer Pages und RSS Feeds.

3.1 Die Datenstrukturen von OpenStreetMap

Der folgende Abschnitt beschäftigt sich mit der Verwaltung von Karten-Rohdaten, d.h. wie werden die von den Mappern gesammelten Informationen intern gespeichert, um von den Anwendungsprogrammierern möglichst einfach benutzt werden zu können (vgl. dazu [4]).

3.1.1 Node

Das kleinste OSM-Objekt, das es gibt, ist der sogenannte Node bzw. Knoten. Knoten repräsentieren GPS-Punkte, wie sie zum Beispiel von einem GPS-Logger erzeugt werden. Ein Node hat immer eine ID, einen Breitengrad und Längengrad, die seine geographische Position angeben, eine Versionsnummer, einen Zeitstempel mit dem letzten Änderungsdatum, eine Changeset[1] ID, die angibt in welchem

[1]Ein Changeset, zu deutsch Änderungssatz, erfasst alle Änderungen, die während einer Sitzung von einem Mapper vorgenommen wurden.

Changeset der Knoten das letzte Mal bearbeitet wurde, und eine User ID mit einem dazugehörigen User. Ein Minimalbeispiel für einen Knoten:

Listing 3.1: Beispiel für einen Node

```
<node id="3578117" lat="48.8851266" lon="12.6169650"
   version="12" timestamp="2013-03-16T21:55:54Z"
   changeset="122312034" uid="1999074" user="perihel"/>
```

Knoten bilden das Gerüst der OSM-Karten. Sie bestimmen den Verlauf von Straßen oder Uferlinien, die Umrisse eines Feldes, markieren Häuserecken, Ampeln oder Briefkästen.

3.1.2 Way

Auf den Knoten bauen die komplexeren Strukturen auf, wie zum Beispiel die Ways bzw. Wege. Ein Way in OSM besteht aus mindestens zwei Knoten und verknüpft diese miteinander. Wie auch der Name schon sagt, werden Ways oder Wege vor allem benutzt, um Straßen zu zeichnen. Wege werden meistens verwendet, um jegliche Art von Verbindung zwischen Punkten herzustellen, zum Beispiel auch Hauswände oder Uferverläufe. Mit Wegen können aber auch Flächen gezeichnet werden, in dem man als Start- und Endpunkt denselben Knoten auswählt. Ein kleines Beispiel für einen typischen Weg ist das folgende:

Listing 3.2: Beispiel für einen Way

```
<way id="9714974" version="16" timestamp="2012-06-28T23
  :41:09Z" changeset="122312034" uid="1999074" user="
  perihel">
    <nd ref="304335953"/>
    <nd ref="1563547869"/>
    <nd ref="1396523373"/>
    <nd ref="13878099"/>
 </way>
```

Wege haben, bis auf das Fehlen der Längen- und Breitengrade, dieselben Metatags (mehr zu Tags in Kürze) wie Knoten. Die einzelnen Knoten, die den Weg definieren, werden über Referenztags eingebunden. Die Referenz `<nd ref="13878099"/>` verweist also auf den Knoten mit der ID 13878099. Bei den OSM-Daten ist zu beachten, dass die ID-Zahlenräume für Knoten, Wege und Relationen getrennt voneinander sind, d.h. zu der ID 13878099 kann es sowohl einen Knoten, einen Weg, als auch eine Relation geben.

Das oben angegebene Beispiel eines Weges führt auf der OSM-Karte zu folgendem Ergebnis (Abb. 3.1):

Abbildung 3.1: OpenStreetMap - Beispiel für einen Way[2]

3.1.3 Relation

Der größte und umfassendste Datentyp in OSM ist die Relation. Mit Relationen werden zum Beispiel die Grenzen der Ozeane erfasst, komplexe Formen, wie die Umrisse des Pentagons [3] oder auch abstrakte Dinge, wie Buslinien oder sogar Abbiegevorschriften. Relationen sind auch der Datentyp, auf dem OSMarelmon arbeitet.
Eine Relation hat, wie ein Way, Referenzen auf andere OSM-Elemente. Allerdings beschränkt sich die Relation nicht nur auf Knoten, sondern kann auch auf Wege oder andere Relationen verweisen. Die Relation ist also ein abstraktes Element in OSM. Eine Straße, wie die Innstraße in Passau würde man als Weg zeichnen, den Donauradwanderweg allerdings als Relation, die vorhandene Wege und Knoten referenziert. Ein kleines (konstruiertes) Beispiel:

Listing 3.3: Beispiel für eine Relation

```
<relation id="1239081" version="2" timestamp
 ="2012-06-28T20:41:04Z" changeset="122312034" uid
 ="1999074" user="perihel">
  <member type="node" ref="867397262" role="stop"/>
  <member type="node" ref="213127" role="platform"/>
  <member type="way" ref="19012912" role=""/>
  <member type="relation" ref="09278124" role=""/>
</relation>
```

[2]©OpenStreetMap.org contributors, Data: ODbL, Map: cc-by-sa

Eine Relation hat dieselben Metatags, wie Ways. Im Gegensatz zu den Wegen werden die Referenzen einer Relation *Member* bzw. Mitglieder genannt und es ist eine Möglichkeit gegeben, den Mitgliedern verschiedene Rollen innerhalb der Relation zuzuweisen. Zum Beispiel hat der erste Knoten die Rolle `role="stop"`, was in diesem Zusammenhang bedeuten soll, dass an diesem Punkt eine Bushaltestelle ist. Der Knoten mit der Rolle `platform` weist auf einen Bussteig hin. Außerdem kann eine Relation auch aus nur einem einzigen Mitglied bestehen.

3.1.4 Tags

Wie schon teilweise in den vorangegangenen Beispielen erwähnt, hat jede OSM-Datenstruktur bestimmte Metatags und von den Benutzern frei handhabbare Tags. Die Metatags finden sich immer in der ersten Zeile nach dem Typ des Objekts: `<typ id=... version=... timestamp=... changeset=... uid=... user=...>`. Sie bilden die sogenannten Metadaten des Objekts. Vor dem Ende der Beschreibung des Objekts, also vor dem End-Tag `</typ>` bietet OSM den Nutzern die Möglichkeit, selbst definierte Tags einzufügen. Somit kann zum Beispiel ein Knoten als Kreuzung ausgezeichnet werden und beschrieben werden, dass der kreuzende Verkehr durch Ampeln geregelt wird [9]: `<tag k="crossing" v="traffic_signals"/> <tag k="highway" v="traffic_signals"/>`. Auch Wege und Relationen können durch solche Tags ausgezeichnet werden: Zum Beispiel die zugelassene Höchstgeschwindigkeit auf einer Straße, die im Winter einen anderen Wert hat: `<tag k="maxspeed:winter" v="60">` [22]. Ein Beispiel für Tags bei Relationen:

Listing 3.4: Beispiel für Tags in OSM

```
<relation id="1239081" version="2" timestamp
 ="2012-06-28T20:41:04Z" changeset="122312034" uid
 ="1999074" user="perihel">
  <member type="node" ref="1934002362" role=""/>
  <member type="node" ref="1934002363" role=""/>
  <member type="node" ref="1934002994" role=""/>
  <member type="node" ref="1934002993" role=""/>
  <tag k="local_ref" v="4;5"/>
  <tag k="network" v="SBV"/>
  <tag k="operator" v="Stadtwerke Straubing"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
</relation>
```

Die Tags bestehen immer aus *key-value*-Paaren. Der Key zeichnet aus, was der Tag beschreiben soll. Im obigen Beispiel macht der `operator`-Key klar, dass dieser Tag den Betreiber der Relation beschreibt: die Stadtwerke Straubing. Die OSM-User haben bei der Vergabe freie Hand und können grundsätzlich neue Tags frei erfinden. Jedoch muss beachtet werden, dass die Programme, die OSM-Daten benutzen, nur allgemein anerkannte und etablierte Tags erkennen können.

3.1.5 OSM Format

Sämtliche OSM-Daten werden im OSM XML Datei Format gespeichert und an die Nutzer weitergegeben [17]. Die bekannteste .osm Datei ist die *planet.osm*, die den gesamten Planet enthält [10]. Grundsätzlich haben die .osm Dateien folgende Struktur (siehe auch Beispiel 3.5):
Das Dokument beginnt mit dem XML Header, der die Kodierung festlegt. Hierauf folgt ein OSM Element, das die Version der API und den Generator, der die Datei erzeugt hat, enthält. Optional kann die OSM-Datei auch ein `<bounds minlat="..." minlon="..." maxlat="..." maxlon="..."/>` enthalten [4]. Die OSM-Objekte werden nun in der folgenden Reihenfolge geliefert: Zuerst ein Block mit allen angeforderten Nodes, dann ein Block, der alle Wege, deren Tags und Referenzen auf die Knoten enthält, und zum Schluss ein Block mit den Relationen, deren Referenzen und Tags. Über die Reihenfolge der Objekte innerhalb der Blocks oder das Vorhandensein eines Blocks dürfen keine Annahmen getroffen werden.
Die Vorteile der .osm Dateien sind dieselben, wie die des XML-Formats [17]: Die Dateien können von Menschen gelesen und verstanden werden und die Verarbeitung der Dateien ist systemunabhängig - das Einzige, was benötigt wird, ist ein Parser. Zudem ist die Kompressionsrate der Dateien relativ hoch. Die Kompression wird allerdings nur durch den Nachteil der Größe der Dateien notwendig: Die aktuelle planet.osm Datei hat unkomprimiert eine Größe von ca. 370 GB (Stand: Februar 2013) [10]. Außerdem benötigt das Parsen der Dateien bei genügend großen Dateien auch mit effizienten Parsern einige Zeit.

Listing 3.5: Beispiel für eine .osm Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API">

<node id="3578117" lat="48.8851266" lon="12.6169650"
   version="12" timestamp="2013-03-16T21:55:54Z"
   changeset="122312034" uid="1999074" user="perihel"/>
 <way id="972974" version="16" timestamp="2012-06-28T23
   :41:09Z" changeset="122312034" uid="1999074" user="
   perihel">
```

```
    <nd ref="304335953"/>
    <nd ref="1563547869"/>
    <nd ref="1396523373"/>
    <nd ref="13878099"/>
  </way>
 <relation id="1239081" version="2" timestamp
   ="2012-06-28T20:41:04Z" changeset="122312034" uid
   ="1999074" user="perihel">
    <member type="node" ref="1934002362" role=""/>
    <member type="node" ref="1934002363" role=""/>
    <member type="node" ref="1934002994" role=""/>
    <member type="node" ref="1934002993" role=""/>
    <tag k="local_ref" v="4;5"/>
    <tag k="network" v="SBV"/>
    <tag k="operator" v="Stadtwerke Straubing"/>
    <tag k="route" v="bus"/>
    <tag k="type" v="route"/>
  </relation>
</osm>
```

3.2 OSMChange-Format und Augmented Diffs

Zusätzlich zu den OSM XML Dateien gibt es auch noch die sogenannten *OSMChange-Dateien* (.osc) und *Augmented Diffs*, wobei letztere eine Erweiterung der OSMChange-Dateien darstellen. Beide Formate dokumentieren die vorgenommenen Änderungen an OSM-Daten [4] und sind dazu gedacht, auf effiziente Art und Weise OSM-Server, die zum Beispiel das planet.osm gespeichert haben, auf den neuesten Stand zu bringen. Ein erneuter Download der planet.osm-Datei und deren Verarbeitung würde Stunden bis Tage in Anspruch nehmen, so dass die Daten zum Zeitpunkt der Fertigstellung des Updates schon wieder veraltet wären. Grundsätzlich liefert der Hauptserver von OSM täglich, stündlich und minütlich erstellte OSMChange-Dateien. Nach [4] haben die täglichen Update-Dateien eine Größe von ca. 0,5% der Planet-Datei, die stündlichen sind meist kleiner als 0,1%. In diesem Abschnitt wird auf die Elemente der OSMChange-Dateien und deren Erweiterung durch die Augmented Diffs eingegangen. Die folgende Beschreibung lehnt sich an die beiden OSM-Wiki Seiten zum OSMChange-Format [8] und zu Augmented Diffs [19] an.

3.2.1 OSMChange Format

OSMChange-Dateien sind ähnlich zu den OSM XML Dateien aufgebaut. Allerdings enthalten sie keine OSM-Elemente, sondern sogenannte *action*-Elemente. Diese Elemente referenzieren wiederum OSM-Objekte und beschreiben die letzte Änderungen, die an diesen vorgenommen wurden. Die OSMChange-Dateien können bis zu drei unterschiedliche action-Typen enthalten: `delete`, `create` und `modify`. Diese Typen beschreiben das Löschen, Erstellen und Verändern von OSM-Elementen. Hier ein kurzes Beispiel einer `create`-action aus [8], das zwei Knoten erzeugt und sie zu einem Weg verbindet:

```
<create>
        <node id="-1" timestamp="2007-01-02T00
   :00:00.0+11:00" lat="-33.9133118622908" lon
   ="151.117335519304">
                <tag k="created_by" v="JOSM"/>
        </node>
        <node id="-2" timestamp="2007-01-02T00
   :00:00.0+11:00" lat="-33.9233118622908" lon
   ="151.117335519304">
                <tag k="created_by" v="JOSM"/>
        </node>
        <way id="-3" timestamp="2007-01-02T00
   :00:00.0+11:00">
                <nd ref="-1"/>
                <nd ref="-2"/>
                <tag k="created_by" v="JOSM"/>
        </way>
</create>
```

Wie man an diesem Beispiel erkennen kann, ist der Weg nicht geo-referenziert. Er hat zwar eine eindeutige geographische Lage, die durch die beiden Knoten definiert ist, der Weg selbst hat aber keine Geo-Referenzierung. Auch Relationen werden ohne diese Information ausgeliefert. Dies macht es für einige OSM-Programme schwieriger [6] nur einen lokalen Ausschnitt zu betrachten. Zum Beispiel gibt es kein Tool, das aus den Diffs lediglich die Informationen für ein bestimmtes Land oder eine bestimmte Stadt filtern kann. Um so ein Feature umsetzen zu können, müssten alle Knoten des Weges in der OSMChange-Datei vorhanden sein. An dieser Stelle setzen die Augmented Diffs an.

3.2.2 Augmented Diffs

Die Augmented Diffs werden von der Overpass API ausgeliefert und statten die Diffs des Haupt-Servers mit Zusatzinformation, wie der Geo-Referenzierung und Metadaten aus. Da Augmented Diffs die für Anwendungsprogrammierer interessanteren Dateien sind, wird im Folgenden auf die action-Typen der Augmented Diffs eingegangen.
Die drei action-Typen `delete`, `create` und `modify` der .osc-Dateien wurden beibehalten und durch einen vierten Typ `info` ergänzt. Für Beispiele der einzelnen action-Typen siehe die Listings 3.6, 3.7, 3.8 und 3.9.

Listing 3.6: Beispiel für Action-type "delete"

```
<action type="delete" waymember="yes">
    <old>
      <node changeset="16608011" id="2351513011" lat
   ="52.2935727" lon="21.0786291" timestamp="2013-06-18
   T19:21:43Z" uid="1999074" user="perihel" version
   ="1"/>
    </old>
    <new>
      <node changeset="16622069" id="2351513011"
   timestamp="2013-06-19T20:45:12Z" uid="1999074" user="
   perihel" version="2" visible="false"/>
    </new>
</action>
```

In Beispiel 3.6 wird ein Knoten, der im Changeset 16608011 das letzte Mal geändert wurde (in diesem Fall sogar: erstellt), gelöscht. Die neue Version des Knotens enthält nun keine geographische Position mehr und der Knoten wurde durch `visible="false"` "unsichtbar" gemacht.

Listing 3.7: Beispiel für Action-type "create"

```
<action type="create" waymember="yes">
    <node changeset="16621835" id="2352672922" lat
   ="36.6162318" lon="4.4786657" timestamp="2013-06-19
   T20:45:06Z" uid="1999074" user="perihel" version
   ="1"/>
 </action>
```

In Beispiel 3.7 wird ein Knoten als Mitglied eines Weges erstellt (`waymember="yes"`).

Listing 3.8: Beispiel für Action-type ”modify”

```
 <action geometry="changed" type="modify" waymember="yes
   ">
     <old>
       <node changeset="16621835" id="2352666971" lat
   ="36.6159173" lon="4.4763656" timestamp="2013-06-19
   T20:38:12Z" uid="1999074" user="perihel" version
   ="1"/>
     </old>
     <new>
       <node changeset="16621835" id="2352666971" lat
   ="36.6159282" lon="4.4763577" timestamp="2013-06-19
   T20:45:06Z" uid="1999074" user="perihel" version
   ="2"/>
     </new>
</action>
```

In Beispiel 3.8 wird die geographische Lage des Knoten geändert: `action geometry="changed" type="modify"`. An diesem Beispiel kann man auch sehen, dass die Änderung in einem einzigen Changeset stattfand.

Listing 3.9: Beispiel für Action-type ”info”

```
<action type="info" waymember="yes">
    <node changeset="13450669" id="293957370" lat
   ="36.7390505" lon="-3.3546916" timestamp="2012-10-11
   T07:21:38Z" uid="1999074" user="perihel" version
   ="3"/>
  </action>
```

Das Beispiel 3.9 zeigt den nur in den Augmented Diffs enthaltenen action-Typ `info`. Dieser action-Typ beinhaltet lediglich OSM-Elemente, die in einer bearbeiteten Relation oder einem bearbeiteten Weg referenziert werden, und dient der vollständigen Erfassung aller von der Änderung betroffenen OSM-Elemente. Dieser action-Typ ist nach [19] optional.

3.3 Overpass API

Die OSM XML Dateien und die Augmented Diffs werden unter anderem von der Overpass API geliefert. Die Overpass API[3] ist eine API, die ausschließlich lesenden Zugriff erlaubt, und dient im Gegensatz zur ursprünglichen Haupt-API dem schnellen Bereitstellen von OSM-Daten [18]. Die Overpass API kann über drei verschiedene Server erreicht werden. Der Server, der für diese Bachelorarbeit verwendet wird, ist über den bereits angegebenen Link erreichbar und hat eine Tageskapazität von ca. 1 Million Anfragen. Damit können derzeit alle Nutzer laut Wiki-Seite ungestört arbeiten, solange 10.000 Anfragen pro Nutzer nicht überschritten werden.
Die Overpass API hat eine eigene Anfragesprache, die Overpass Query Language (Overpass QL) [18]. Diese Anfragesprache erlaubt den Benutzern eine Selektion der Daten durch die Angabe von Tags oder einer Bounding Box. Alternativ zur Overpass QL können Anfragen auch mittels XML an die API gestellt werden. Die beiden Anfragesprachen können durch ein Webformular[4] ineinander umgewandelt werden. Im folgenden wird eine Query im XML Schema erstellt und erklärt, in die Overpass QL übersetzt und die Verwendung in OSMarelmon beschrieben.
Als Beispiel werden wieder die Buslinien der Stadt Passau verwendet. Für den restlichen Abschnitt vergleiche dazu auch [18]. Um eine XML Anfrage zu erstellen, wird der Start-Tag `<osm-script>` benötigt. Da die Buslinien allesamt als Relationen erfasst sind, muss die Anfrage nach Relationen suchen. Weiter muss nun das Ergebnis eingeschränkt werden: Dafür werden die `<has-kv>` Tags benötigt, wobei *has-kv* für *has key value*, also *hat Schlüssel Wert*, steht. Um die für das Beispiel geforderten Buslinien in Passau zu erhalten, reicht es die key-value-Paare, `operator = Stadtwerke Passau` und `line = bus` zu verwenden (siehe Bsp.: 3.10). Der Tag `<print mode="meta"/>` fordert die Overpass API dazu auf, auch die Metadaten für die einzelnen Objekte mitzuliefern.

Listing 3.10: Beispiel für eine XML Anfrage

```
<osm-script>
  <query type="relation">
    <has-kv k="operator" v="Stadtwerke Passau"/>
    <has-kv k="line" v="bus"/>
  </query>
<print mode="meta"/>
</osm-script>
```

[3] Homepage: `http://overpass-api.de/`
[4] `http://overpass-api.de/query_form.html`

Auf der Website "overpass turbo"[5] kann die Anfrage eingegeben, ausgeführt und das Ergebnis betrachtet werden. Als Ergebnis liefert die Overpass API jetzt alle Relationen, die Buslinien in Passau darstellen. Allerdings liefert die Anfrage keine Knoten oder Wege. Um auch diese Elemente zu erhalten, wird das *union* Statement verwendet (siehe Bsp.: 3.11):

Listing 3.11: Beispiel für eine XML Anfrage, die rekursiv alle Knoten und Wege der Relationen einsammelt

```
<osm-script>
  <union>
  <query type="relation">
    <has-kv k="operator" v="Stadtwerke Passau"/>
    <has-kv k="line" v="bus"/>
  </query>
  <recurse type="relation-node" into="nodes"/>
  <recurse type="relation-way"/>
  <recurse type="way-node"/>
</union>
<print mode="meta"/>
</osm-script>
```

Das *union* Statement vereinigt die Menge der Relationen, die wie in der ersten Anfrage gesucht wurden, mit der Menge der rekursiv erhaltenen Daten. Durch den rekursiven Aufruf `<recurse type="relation-node" into="nodes"/>`, werden dem Dokument auch die Knoten, die in einer Relation referenziert werden, hinzugefügt. Analog funktioniert `<recurse type="relation-way"/>`. `<recurse type="way-node"/>` fügt letztendlich noch alle in den Wegen referenzierten Knoten hinzu.
OSMarelmon verwendet jedoch nicht das XML Format für seine Anfragen, sondern die Overpass QL. Somit kann das Programm den Vorteil nutzen, die Anfragen über HTTP-GET Anfragen in die URL's zu kodieren. Wie oben bereits erwähnt, können XML Anfragen und Overpass QL Anfragen automatisch über ein Webformular ineinander umgewandelt werden. Die äquivalente Overpass QL Query zu obigem Beispiel würde dann wie in Beispiel 3.12 aussehen. Das Webformular unterscheidet zwischen kompakter und "schöner" QL, wobei der Unterschied lediglich in fehlenden Leerzeichen und Zeilenumbrüchen liegt. Die URL zum Download der Daten dieser Anfrage sieht folgendermaßen aus: `overpass-api.de/api/interpreter?data=(relation["operator"="Stadtwerke Passau"]["line"="bus"];node(r)->.nodes;way(r);node(w););out meta;`

[5] Homepage: `http://overpass-turbo.eu/`

Listing 3.12: Overpass QL Query

```
(
  relation
    ["operator"="Stadtwerke Passau"]
    ["line"="bus"];
  node(r)->.nodes;
  way(r);
  node(w);
);
out meta;
```

3.4 Suchfunktion

Nachdem nun bereits ein Weg geschildert wurde, nach OSM-Daten zu suchen, wird in diesem Abschnitt gezeigt, welche Suchfunktionalität man in einem Programm anbieten könnte, um möglichst viele Benutzerwünsche zu befriedigen. Dazu wird beispielhaft die äußerst umfangreiche und mächtige Suche des Editors *JOSM*[6] erklärt.

3.4.1 JOSM

JOSM gehört zur Programmklasse der Editoren von OSM. Mit Hilfe von JOSM können Mapper OSM-Daten bearbeiten und auf die OSM-Datenbank laden. Es gibt weitere Editoren für OSM-Daten, wie zum Beispiel den Online Editor *Potlatch* oder *Merkaartor* [12]. Um einen kurzen Überblick über die Grundfunktionen von JOSM zu geben, folgt noch eine sehr knappe Beschreibung, wie ein Mapper vorgehen kann, wenn er ein neues Gebiet in OSM erfassen will (vgl. [4]):
Zuerst lädt sich der Mapper die Daten seines GPS-Geräts auf den PC (meist in Form von .gpx-Dateien). Nach dem Start von JOSM kann er diese importieren. Die erstellten GPS-Punkte werden nun in JOSM sichtbar. Jetzt muss der Mapper noch das ausgewählte Gebiet, das er vervollständigen will, vom OSM-Server laden. Hierauf kann der Mapper die Daten, die er erfasst hat, in OSM-Elemente umwandeln und die Änderungen wieder auf den OSM-Server laden.

3.4.2 Die JOSM Suche

Doch natürlich können mit JOSM auch bereits erfasste OSM-Elemente, wie z.B. Buslinien bearbeitet werden. Dafür muss zuerst wieder ein Gebiet vom OSM-Server

[6] Homepage: `http://josm.openstreetmap.de/`

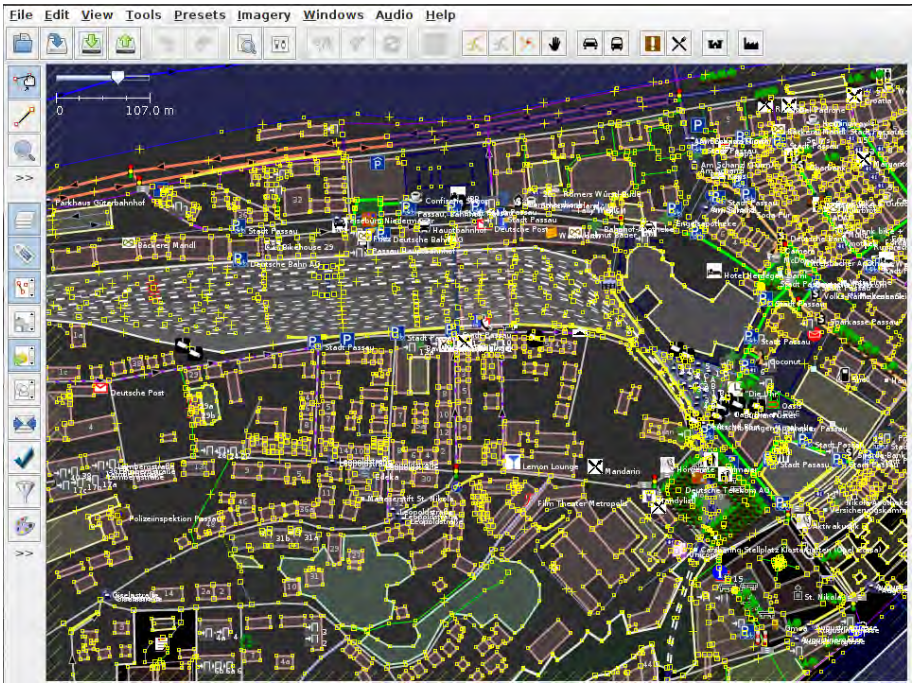

Abbildung 3.2: Screenshot von JOSM - Ausschnitt vom Passauer Hbf und Zentraler Omnibusbahnhof

geladen werden. Als Beispiel wird hier die Gegend um den Passauer Haupt- und Omnibusbahnhof verwendet. Doch um in diesem Gewirr aus GPS-Punkten, das die Bearbeitungsansicht liefert (vgl. Abb. 3.2), etwas zu finden, wird eine Suchfunktion benötigt. Diese Suche wird im Folgenden erläutert.
Zugriff auf die Suchfunktion erhält man über das Menü am oberen Rand: "Edit" → "Search...". Es öffnet sich der Dialog aus Abb. 3.3. Der Dialog selbst ist minimalistisch gehalten, es existiert lediglich ein Eingabefeld und ein paar Buttons, um zu definieren, wie mit dem Suchergebnis verfahren werden soll.

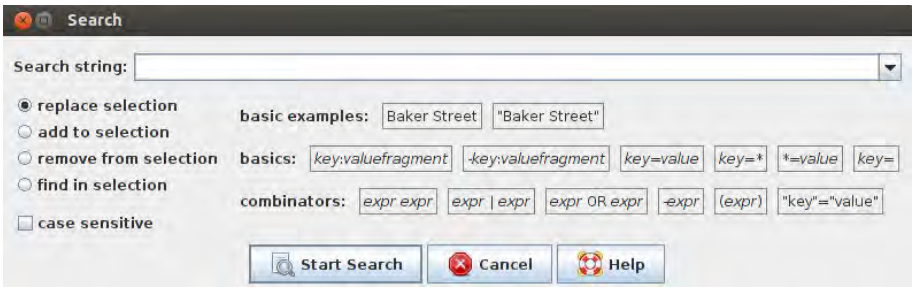

Abbildung 3.3: Screenshot von JOSM - Suchdialog

Doch von diesem kleinen Fenster darf man sich nicht täuschen lassen, denn JOSM bietet eine äußerst umfangreiche und mächtige Suche an. Im OSM-Wiki existiert eigens eine Seite, die sich nur mit dieser Suchfunktion befasst [13]. Sämtliche in diesem Unterabschnitt folgende Erklärungen und Beispiele stammen von dieser Seite.
Nachdem ein Suchterm eingegeben und auf "Start Search" geklickt wurde, passiert folgendes: JOSM baut aus dem Suchausdruck eine Baumstruktur auf. Anschließend wird dieser Baum zur Suche verwendet. Es gibt verschiedene Basis-Schlüsselwörter, welche zu komplexeren Suchanfragen kombiniert werden können. Ein solches Basis-Schlüsselwort stellt unter anderem `type:relation` dar (wobei `relation` auch durch `way` oder `node` ersetzt werden könnte). Sucht man nun nach `type:relation`, liefert die Suche alle Relationen im geladenen Gebiet zurück und färbt diese in der Bearbeitungsansicht ein.
Zusätzlich zu den Basis-Schlüsselwörtern gibt es sogenannte *Modifier*. Im Moment ist das genau ein einziger, nämlich das "-" Zeichen als das logische NICHT. Dieses Zeichen invertiert Anfragen und ist nur an das nächste Schlüsselwort der Anfrage gebunden. Somit liefert `-type:relation` alle OSM-Elemente, die keine Relation sind.
Wie anfangs erwähnt, können die Basis-Schlüsselwörter kombiniert werden. Es gibt zwei Arten, auf die Anfragen kombiniert werden können. Im Suchdialog werden zwar ganze sechs Kombinatoren angeboten, allerdings werden dort zwei semantisch äquivalente Schreibweisen, das logische NICHT, Klammerung und das Escapen von Anführungszeichen hinzugenommen. Man kann sie mit dem logischen UND (es muss kein explizites Schlüsselwort angegeben werden, die Anfragen werden einfach aufgezählt) oder dem logischen ODER (Schlüsselwort `OR` oder `|`) verknüpfen. Das ODER bindet in der JOSM-Suche stärker als das UND und ist inklusiv. Ein kurzes Beispiel für die Verknüpfung: `-type:relation user:Peda` liefert alle OSM-Elemente zurück, die keine Relation sind und vom Benutzer "Peda" geändert wurden. `type:relation OR user:Peda` findet alle Elemente, die eine Relation sind oder vom Benutzer "Peda" bearbeitet wurden.
Natürlich gibt es für die Suche weitere Schlüsselwörter und Beispiele, die auf der Wiki-Seite nachgelesen werden können. Im Zusammenhang mit dieser Arbeit sind sie jedoch nicht mehr relevant, es sollte lediglich gezeigt werden, welch umfassende Suchfunktionen man in OSMarelmon hätte einbauen können. Allerdings wurde erkannt, dass eine zur JOSM-Suche ähnliche Suchfunktion für die Programmzwecke vollkommen überdimensioniert wäre, weshalb eine einfachere Suche implementiert wurde, die nach Relationen, wie in Abschnitt 3.3 beschrieben, sucht.

3.5 Webtechnologie

Da das Programm zu dieser Bachelorarbeit eine Webanwendung ist, wurden Websites zur Interaktion mit den Benutzern benötigt. Die Programmierung dieser Websites wurde mit Hilfe der JavaServer Pages (JSP) Technologie umgesetzt[7], da durch den Einsatz der JSP Technologie und Java Servlets die Applikation weiter komplett plattformunabhängig gehalten und eine strikte Trennung zwischen Programmlogik und Nutzeransicht erreicht wurde.

3.5.1 Servlets

Servlets sind spezielle Java Klassen, die bestimmte Methoden zur Vearbeitung von HTTP-Anfragen anbieten müssen. Servlets werden vom Webserver instanziiert und nehmen eingehende Anfragen von Clients, wie z.B. einem Browser entgegen. Die Anfrage wird daraufhin bearbeitet, eine Antwort erstellt und an den Client zurückgeschickt.
Wie die Verarbeitung einer Anfrage aussieht, kann in Beispiel 3.13 betrachtet werden.

Listing 3.13: Auszug der handleRequest Methode aus OSMarelmon

```
1  private void handleRequest(HttpServletRequest req,
       HttpServletResponse res) throws ServletException,
       IOException {
2  ...
3  String actionID = getActionId(req, ACTIONID_START);
4  if (actionID.equals(ACTIONID_START)) {
5          action = new StartAction();
6  } else if (actionID.equals(ACTIONID_ADD)) {
7          action = new AddRelationAction();
8  }
9  ...
10 else {
11         action = new ErrorAction("Error!", null);
12 }
13 nextSite = action.execute(req);
14 req.getRequestDispatcher(nextSite).forward(req, res);
15 }
```

In diesem Beispiel wird die Methode `handleRequest` abgebildet, welche eine zentrale Rolle in der Bearbeitung der Anfragen darstellt. Als Parameter erhält die

[7] `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`

Methode die Anfrage des Clients (`req`) und das Antwort-Objekt (`res`). Das `req`-Objekt enthält unter anderem einen sogenannten Action-Parameter, der dem Programm mitteilt, auf welchen Button der Benutzer der Website geklickt hat. Entsprechend dieses Parameters werden unterschiedliche Teile des Programms ausgeführt. Am Ende der Methode wird der Benutzer, je nach Resultat der durchgeführten Aktion, auf eine andere Seite weitergeleitet.

3.5.2 JSP Seiten

JSP Seiten sind Textdokumente, ähnlich zu HTML Seiten. Sie können nicht nur JSP Code enthalten, sondern auch HTML oder Java. Mit Hilfe von JSP können Websiten auf einfache Weise entworfen und gestaltet werden. Zudem erleichtern JSP Seiten die Verknüpfung der Benutzeransicht mit der Programmlogik. Ein Minimalbeispiel einer JSP Seite kann in Beispiel 3.14 in betrachtet werden.

Listing 3.14: Beispiel einer JSP Seite, aus [2]

```
1  <%@ page import="java.util.*" %>
2  <% GregorianCalendar clock = new GregorianCalendar(); %>
3  <html>
4    <head>
5      <title>Clock</title>
6    </head>
7    <body>
8      Die aktuelle Zeit ist: <%= clock.getTime() %> <br/>
9      Der Server steht in Zeitzone <%= clock.getTimeZone()
       %>
10   </body>
11 </html>
```

In diesem Beispiel wird in der ersten Zeile mit der *Direktive* `<%@ page ... %>` eine Java-Bibliothek eingebunden. Tags, die ein Prozentzeichen am Anfang und Ende haben (`<% ... %>`), werden *Scriptlets* genannt und enthalten Java Code. Grundsätzlich sollte der Einsatz von Scriptlets aber vermieden werden, da für diese Zwecke die *JSP Standard Tag Library* (JSTL) und die *JSP Expression Language* entwickelt wurden. Im Beispiel folgt nun der HTML Rumpf, in welchem beliebig JSP Tags und Scriptlets verwendet werden dürfen. Dieses kurze Beispiel würde eine Website erzeugen, die dem Benutzer die aktuelle Uhrzeit und die Zeitzone, in der der Server steht, ausgibt. Die Besonderheit an JSP Seiten ist, dass sie einen Compiler durchlaufen und in Servlets umgewandelt werden.
In OSMarelmon wurden die JSP Seiten mit den Servlets zu einer Model-View-Controller (MVC) Architektur kombiniert (siehe dazu Abb. 3.4). Das MVC-

Konzept dient in der Programmierung zur Trennung der Programmlogik (Model) von der Benutzeransicht (View) und der Steuerung des Programms (Controller). Somit kann das Model bzw. die View bei Beibehaltung der Schnittstelle auf einfache Weise ausgetauscht oder verändert werden, ohne dass Änderungen am jeweils anderen Teil vorgenommen werden müssen.

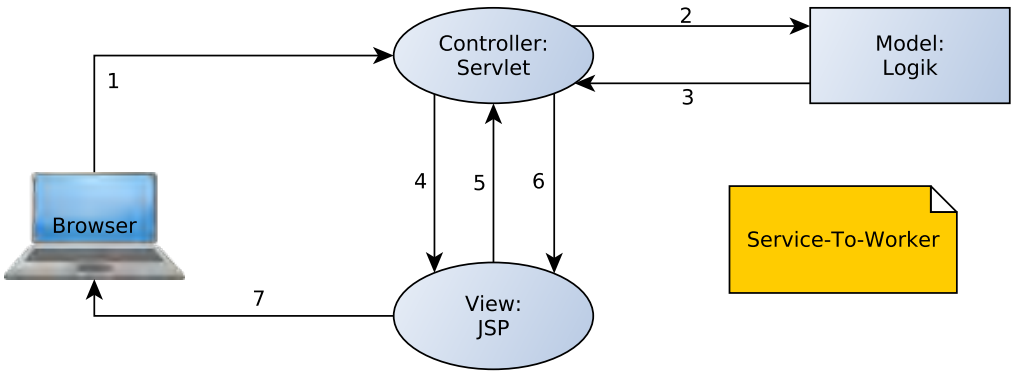

Abbildung 3.4: JSP, Servlet und Logik in der MVC-Architektur, Bild angelehnt an [2]

In der MVC-Architektur übernimmt das Servlet die Controller-Funktion, nimmt die Anfrage eines Clients entgegen und ruft die Logik des Programms auf. Nach Berechnung des Ergebnisses im Model erhält wieder das Servlet die Kontrolle und wählt die entsprechende JSP Seite aus. Daraufhin holt sich die JSP Seite die benötigten Daten vom Servlet und gibt sich selbst als Antwort an den Client zurück. Dieser Ablauf wird auch als Service-To-Worker bezeichnet [2].
Insgesamt werden in OSMarelmon vier JSP Seiten in Zusammenarbeit mit zwei Servlets verwendet: Die Start JSP Seite, die der Benutzer sieht, wenn er das Programm mit seinem Browser aufruft, jeweils eine JSP Seite, die den (Miss-)Erfolg des Hinzufügens einer Anfrage anzeigen, und eine JSP Seite, die sich selbst neu lädt und dem Nutzer anzeigt, das das Hinzufügen seiner Relationen in Bearbeitung ist. Das eine der beiden Servlets ist das Controller-Servlet, dass das Hinzufügen von Relationen regelt, das zweite ist für die Anzeige der RSS-Feeds zuständig.

3.6 RSS

Dieser Abschnitt geht auf RSS ein, erklärt, was unter RSS verstanden wird und in welchem Zusammenhang es verwendet wird. Zudem wird der Aufbau eines RSS-Dokuments erläutert und die Veröffentlichung eines RSS-Feeds beschrieben.

3.6.1 Erklärung und Verwendung

RSS steht für *Really Simple Syndication*, beschreibt einen einfachen Weg, Schlagzeilen oder Inhalte über das Internet zu verteilen und ist in XML geschrieben. Die aktuelle Version ist RSS 2.0. Durch die Verwendung von RSS Feeds können Nutzer mit Hilfe von RSS Aggregatoren (Programme, die abonnierte RSS Feeds verwaltet) die Websiten auf neue Inhalte überprüfen ohne diese besuchen zu müssen. Somit ist RSS vor allem für Nachrichtenseiten, Firmen oder Kalender nützlich [7]. Auch für das Programm dieser Bachelorarbeit ist RSS von großem Nutzen, weswegen im folgenden näher darauf eingegangen wird.
RSS funktioniert so, dass auf einem Server, auf dem auch die Website gehostet wird, gleichzeitig einer oder mehrere RSS Feeds gespeichert sind. Der Nutzer gelangt durch einen Link auf einen RSS Feed und kann diesen nun abonnieren. Ein typischer RSS Feed wird in Abb. 3.5 dargestellt. Der RSS Feed kann nun mit einem Feed Aggregator abonniert werden, so dass der Nutzer nicht mehr die Website aufrufen muss, um über Neuigkeiten informiert zu werden, sondern lediglich den Feed Aggregator.

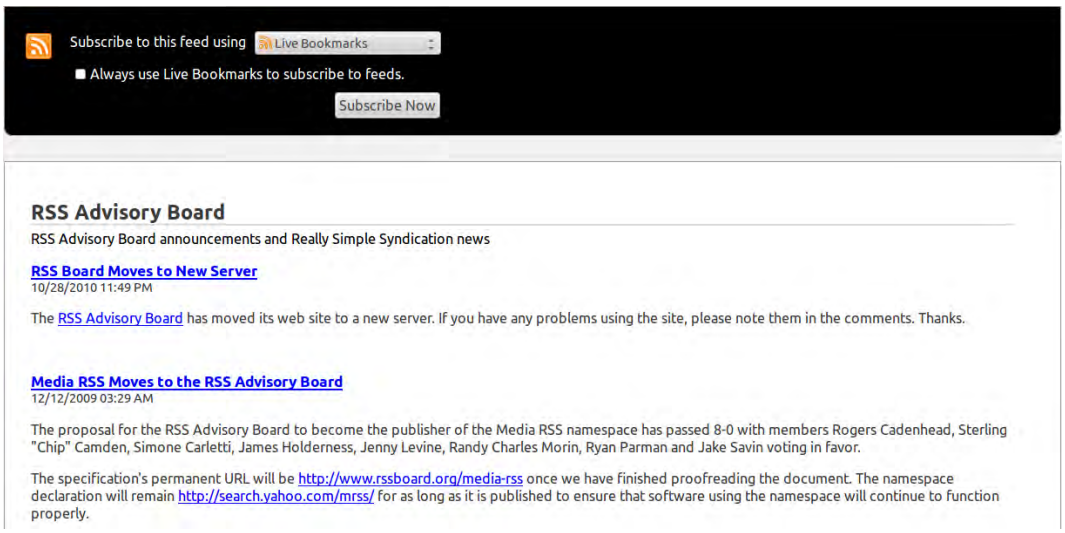

Abbildung 3.5: Typische RSS Seite von `http://feeds.rssboard.org/rssboard?format=xml`

3.6.2 Aufbau eines RSS Dokuments

Ein RSS Dokument ist, wie eingangs erwähnt, in XML geschrieben. Die erste Zeile eines RSS Dokuments beschreibt daher die verwendete XML Version und die Kodierung. Darauf folgt das RSS Item mit der RSS Version. In der dritten Zeile beginnt ein *Channel* Objekt, das den eigentlichen Feed definiert. Das Channel-Element besitzt drei notwendige Kind-Elemente: `<title>` (legt den Namen des Feeds fest), `<link>` (Link zum RSS Feed) und `<description>` (beschreibt den Channel). Darüber hinaus kann jedes Channel-Element ein oder mehrere *Item* Elemente beinhalten, wobei diese Items die RSS Nachrichten darstellen. Jedes Item hat wiederum dieselben vorgeschriebenen Kind-Elemente, wie ein Channel [7]. Der folgende Code beschreibt ein Beispiel-RSS Dokument:

Listing 3.15: Beispiel eines RSS Dokuments aus [7]

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</
   description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>http://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</
   description>
  </item>
</channel>

</rss>
```

Um den Text einer RSS Nachricht formatieren zu können, werden HTML Tags benötigt. Bei der Verwendung von reinem HTML Code in der `description` eines Items oder Channels, werfen die Parser der Feed Aggregatoren einen Fehler.

Um dies zu vermeiden muss der HTML Code mittels `<![CDATA[ ... ]]>` escapt werden.

3.6.3 Veröffentlichung des Feeds

Um RSS Feeds zu veröffentlichen, bestehen zwei Möglichkeiten: Zum einen kann der Feed bei bestehenden Feed Verzeichnissen registriert und veröffentlicht werden, zum anderen kann man seine Feeds selbst managen. OSMarelmon wurde so programmiert, dass die Feeds selbst verwaltet werden. Ein ausschlaggebender Grund hierfür war die dynamische Erzeugung von RSS Feeds: Sobald eine neue Anfrage überwacht wird, wird ein neuer Feed erstellt, was potenziell zu sehr vielen Feeds führen kann. Zur Verwaltung der Feeds wurde *ROME* verwendet [5]. ROME bietet eine Open Source Java Bibliothek zur Verwaltung von RSS und Atom Feeds an und unterstützt alle gängigen Formate. Alle von OSMarelmon erzeugten RSS Feeds werden mit Hilfe von ROME in der Version 2.0 veröffentlicht. Beispiele für die erzeugten Feeds können in den Abbildungen 3.6 und 4.3 betrachtet werden.

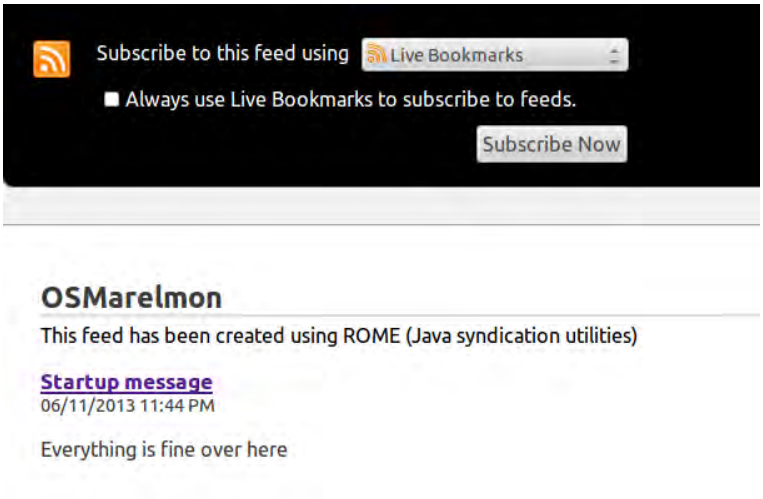

Abbildung 3.6: Von OSMarelmon erzeugter RSS Feed

Kapitel 4

OSMarelmon - Der OpenStreetMap Relation Monitor

In diesem Kapitel wird das Programm zu dieser Bachelorarbeit besprochen. Nach einer Vorstellung der Features, wird der Algorithmus, der das Überprüfen von Relationen durchführt, und der Programmfluss erläutert. Da es bereits Programme gibt, die eine ähnliche Funktionalität wie OSMarelmon anbieten, werden im Anschluss einige Programme kurz vorgestellt und mit OSMarelmon verglichen. Im letzten Abschnitt werden sinnvolle Erweiterungsmöglichkeiten für das Programm diskutiert.

4.1 Programmvorstellung

In diesem Abschnitt wird ein Überblick über OSMarelmon gegeben. Zum einen werden die eingebauten Funktionen erklärt, zum anderen werden die programminternen Datenstrukturen und deren Verwendung erläutert. Zudem wird die softwaretechnische Umsetzung dargelegt.

4.1.1 Funktionen

Die Hauptfunktionalität von OSMarelmon liegt in der Überwachung von OSM-Relationen. Der Benutzer kann über das Webinterface eine Overpass Query erstellen (vgl. auch Abb. 4.1).

Durch einen Klick auf "Add relation to monitor" erstellt OSMarelmon nach der Validierung der Eingabefelder aus den eingegebenen key-value-Paaren eine Overpass Query. Die erzeugte Query wird dann an die Overpass API weitergeleitet. OSMarelmon lädt sich das Resultat der Anfrage herunter und parst das Ergebnis. Der Benutzer wird über den Status seiner Anfrage ständig auf dem Laufenden

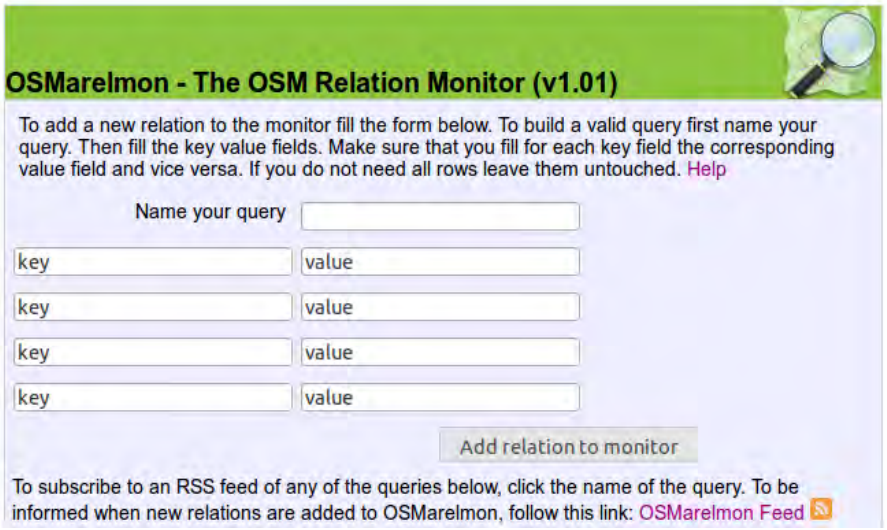

Abbildung 4.1: OSMarelmon - Overpass Query Formular

gehalten. Falls das Ergebnis seiner Anfrage leer ist oder andere Fehler auftraten, wird er zu einer Fehlerseite weitergeleitet. Andernfalls, bekommt der Nutzer eine Erfolgsmeldung zu sehen. Wurde die Anfrage in OSMarelmon gespeichert, kann der Benutzer jetzt auf einen entsprechenden RSS Feed zugreifen (siehe Abb. 4.2).

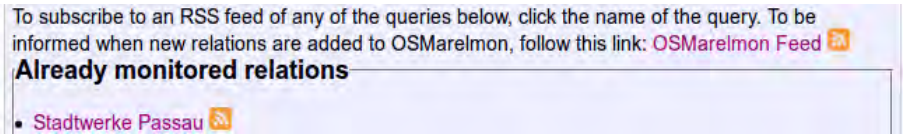

Abbildung 4.2: OSMarelmon - Links zu den entsprechenden RSS Feeds in den Namen der Anfragen

OSMarelmon führt nun periodisch Überprüfungen der Anfrage durch und meldet Änderungen an interessierte Benutzer, die den entsprechenden RSS Feed abonniert haben. Ein Beispiel für eine solche Änderungsmeldung ist in Abb. 4.3 zu sehen.
Zusätzlich zu den RSS Feeds für jede Anfrage, gibt es auch noch einen Standard Feed für OSMarelmon selbst. Dieser Feed benachrichtigt Abonnenten über neue Relationen, die zu OSMarelmon hinzugefügt wurden.

4.1.2 Softwaretechnik

In diesem Abschnitt wird auf die interne Arbeitsweise eingegangen, d.h. welche Datenstrukturen OSMarelmon zur Verwaltung der Anfragen verwendet, wie die

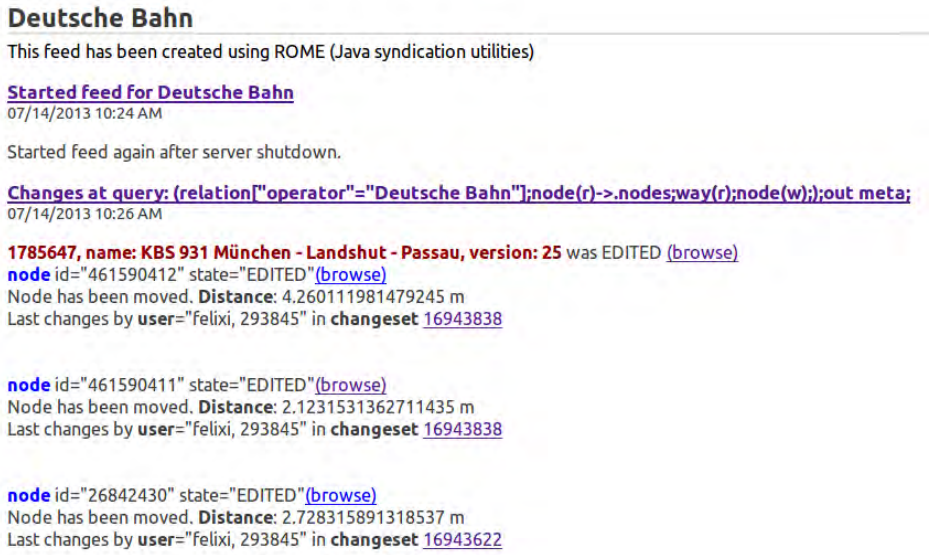

Abbildung 4.3: OSMarelmon RSS Feed - Beispiel für die Anzeige für Änderungen

Überprüfung der Relationen angestoßen wird und auf welche Weise Benutzerinteraktionen behandelt werden.
Da OSMarelmon als Webanwendung auf einem Server auf gleichzeitige Benutzung durch mehrere Nutzer ausgelegt ist, musste die Anwendung thread-safe gestaltet werden. Dies wurde mittels *Java Monitor* Konzept durchgeführt [1]. Ausgangspunkt für dieses Konzept ist eine kritische Ressource und mehrere Threads, die Zugriff auf diese Ressource benötigen. Durch Verwendung des Java Schlüsselworts `synchronized` erhält immer genau ein Thread Zugriff auf die Ressource. Der nächste Thread erhält erst bei Verlassen des ersten Threads Zugriff darauf.
Dieses Konzept wurde mehrmals in OSMarelmon verwendet um fehlerhafte Schreibzugriffe zu vermeiden. Zum einen können Overpass Queries nur sequentiell an die Overpass API gestellt und das Ergebnis heruntergeladen werden, zum anderen ist es nicht erlaubt, zwei Objekte aus OSMarelmon gleichzeitig zu serialisieren.
Im Programm werden mehrere Threads verwendet bzw. erstellt: Im Hintergrund und von Anfang an läuft der sogenannte *CheckThread*. Dieser Thread ist für das Überprüfen der Relationen zuständig. Der Thread liegt - per default - einen halben Tag lang schlafen und stößt dann die Überprüfung der gespeicherten Relationen an. Die zweite Thread-Art ist der *AddThread*. Dieser Thread wird für jede Anfrage erstellt, die ein Benutzer an OSMarelmon schickt. Bei den AddThreads liegt jedoch eine Besonderheit vor: sie registrieren sich beim sogenannten *ThreadManager* und teilen diesem ihren aktuellen Status mit. Dies dient der Benutzerinformati-

on: Möchte der Nutzer im Webinterface eine neue Relation hinzufügen, kann dies unter Umständen bei arbeitendem CheckThread oder großem Downloadvolumen einige Zeit beanspruchen, in der der Nutzer nicht weiß, ob seine Anfrage erfolgreich verabeitet wurde. Der AddThread registriert sich bei seiner Erstellung beim ThreadManager mit einem gewissen Status. Die Seite, die der Benutzer in der Zwischenzeit in seinem Browser angezeigt bekommt (Abb. 4.4), aktualisiert sich in kurzen Abständen und erfragt beim ThreadManager den Status des eigenen AddThreads. Sobald dieser abgearbeitet wurde, erhält der Benutzer dann die Mitteilung über Erfolg oder Misserfolg.

Abbildung 4.4: OSMarelmon - Die Anfrage des Nutzers wird bearbeitet

Der AddThread erhält von der Overpass API eine OSM XML Datei, die dem Parser übergeben wird. Der Parser erstellt aus der Datei ein Java Objekt, mit dem die OSM-Datenstrukturen performant zugreifbar gemacht werden. Bei einer nicht-leeren Datei, d.h. bei einer Datei, die mindestens eine Relation enthält, wird das erstellte Objekt als *.myosm* Datei auf der Festplatte serialisiert und erst bei Überprüfung durch den CheckThread wieder in den Arbeitsspeicher geladen. Zugleich wird nach erfolgreichem Speichern der Anfrage ein RSS Feed erstellt und die Benutzer über den OSMarelmon Standard Feed über die neue Relation benachrichtigt.

4.2 Verwendeter Algorithmus

Dieser Abschnitt beantwortet vor allem die Frage, wie OSMarelmon Änderungen an Relationen, trotz riesiger Datenmengen, effizient bemerkt. Außerdem wird auf Probleme bei der Entwicklung des Algorithmus und mögliche andere Herangehensweisen eingegangen.

4.2.1 Funktionsweise

Wie in Abschnitt 4.1 beschrieben, werden die OSM XML Dateien als serialisierte Java Objekte gespeichert. Wird die Überprüfung durch den CheckThread angestoßen, lädt sich OSMarelmon nacheinander die gespeicherten Dateien in den Arbeitsspeicher. Zu jeder vorhandenen Datei wurde auch die Query mitabgespeichert, so dass sich das Programm die neueste Version von der Overpass API herunterladen kann. Auch diese Version wird geparst, so dass die effizienten Java Datenstrukturen benutzt werden können. Der am häufigsten verwendete Typ ist die Java `HashMap<key, value>` mit der ID des OSM-Elements als Schlüssel und dem Element selbst als Wert. Jedes myosm-Objekt besitzt eine HashMap für jeden OSM-Datentyp. Auf diesen Datenstrukturen beginnt der Algorithmus zu arbeiten. Zur Veranschaulichung des Algorithmus siehe auch Abb. 4.5.

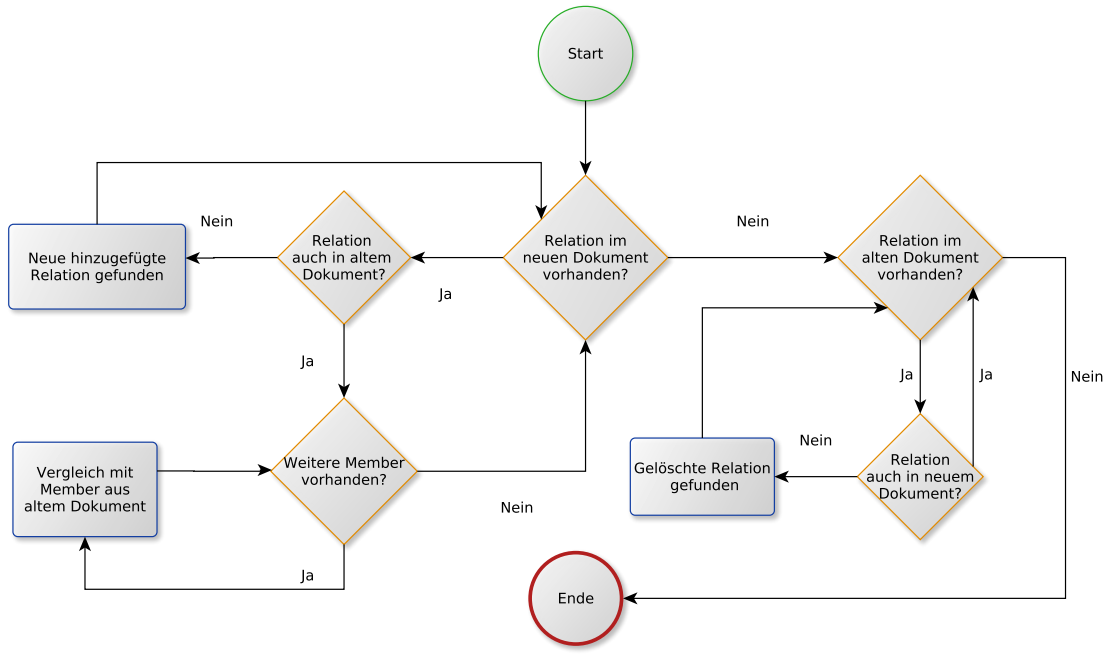

Abbildung 4.5: Ablaufdiagramm des Überprüfungsalgorithmus

Als Eingabe erhält der Algorithmus zwei myosm-Objekte, die gespeicherte, alte und die entsprechende neue Version. Gestartet wird, indem mit einer `for`-Schleife über die Relationen des neuen Dokuments iteriert wird. Für jede Relation wird überprüft, ob dieselbe Relation auch im alten Dokument vorhanden war. Ist dies nicht der Fall, wurde die Relation aus dem Dokument entfernt. Falls die Relation im alten Objekt vorhanden ist, startet eine weitere `for`-Schleife, die durch die Mit-

glieder der Relation läuft und deren OSM-Typen prüft. Dabei werden für jeden OSM-Typ spezifische Überpüfungsmethoden aufgerufen. Diese Methoden vergleichen Tags, Metadaten und Vorhandensein der Objekte in den beiden Versionen der Dokumente. Für jedes OSM-Objekt, für das eine Änderung festgestellt wurde, wird ein entsprechendes `Changed`-Objekt erstellt. Diese Changed-Datentypen speichern alle Informationen, die später auch an den Benutzer geliefert werden. Die Changed-Objekte werden vom Algorithmus gespeichert und später gesammelt in einer Liste von `ChangedRelations` zurückgegeben. Um bei den `Changed`-Objekten den Überblick zu behalten, welches Objekt gelöscht, hinzugefügt oder bearbeitet wurde, existiert das `enum State`, das unter anderem die Werte `REMOVED, ADDED` oder `EDITED` annehmen kann. Nachdem die Relation auf neue oder bearbeitete Mitglieder durchsucht wurde, endet die innere `for`-Schleife. An ihre Stelle tritt eine weitere `for`-Schleife, die dieses Mal über die alte Version der Relation läuft und nach gelöschten Mitgliedern sucht.
An diesem Punkt endet sowohl die innere, als auch die äußere `for`-Schleife. Zu guter Letzt startet eine `for`-Schleife, die alle Relationen des alten Dokuments durchläuft und auf gelöschte Relationen überprüft.

4.2.2 Probleme

Da der Algorithmus die Überprüfung in den zeitlichen Abständen durchführt, die der CheckThread vorgibt, kann es sein, dass manche Änderungen nicht von OSMarelmon erfasst werden. Grundsätzlich wird die OSM-Datenbank minütlich aktualisiert [4], so dass es möglich ist, dass Änderungen nicht an den Abonnenten des RSS Feeds weitergegeben werden können, falls dasselbe OSM-Objekt in der Zeit zwischen den Überprüfungen mehrmals geändert wurde. Aufgrund der seltenen Updates bei getesteten Relationen und um Rücksicht auf andere Nutzer der Overpass API zu nehmen, wird dies in Kauf genommen.

4.3 Bewertung des Algorithmus

Dieser Abschnitt befasst sich mit der Bewertung des Algorithmus. Zum einen wird der Algorithmus bezüglich Ressourcenverbrauch bewertet, zum anderen wird er mit anderen möglichen Herangehensweisen verglichen.

4.3.1 Ressourcenverbrauch

Der Ressourcenverbrauch eines Algorithmus kann in vielerlei Hinsicht gemessen werden. In diesem Abschnitt liegt der Fokus vor allem auf der Geschwindigkeit. Da der Speicherverbrauch eher nebensächlich ist, wird auf diesen nur kurz

eingegangen. Zuerst zur Geschwindigkeit: die obere Schranke für die Laufzeit des Algorithmus lässt sich relativ einfach bestimmen. Sei $r \in \mathbb{N}$ die Anzahl der Relationen und $o \in \mathbb{N}$ die Anzahl der OSM-Objekte in einer OSM XML Datei. Dann gilt zum einen $o \geq r$ und zum anderen lässt sich die Anzahl der Mitglieder einer beliebigen Relation des Dokuments durch o nach oben abschätzen. Hätte eine Relation mehr als o Mitglieder, würden Mitglieder im Dokument fehlen, was aber nicht sein kann, da die Anfrage an die Overpass API alle Mitglieder liefert. Somit lässt sich die Laufzeit mit der O-Notation abschätzen: Laufzeit $T(r) = (r \cdot o) + r \leq (r \cdot r) + r \in O(r^2)$.
Somit würde der Algorithmus ungefähr quadratische Zeit benötigen, um ein Dokument zu überprüfen, wobei erwähnt werden muss, dass der Vergleich von zwei Ways grundsätzlich auf dieselbe Weise stattfindet wie der Vergleich von Relationen. Dies würde in der O-Notation allerdings nicht ins Gewicht fallen.
Bei realen Dokumenten fällt jedoch auf, dass die Abschätzung der Mitglieder einer Relation durch die OSM Objekte eines Dokuments in der Regel viel zu grob ist. Für drei verschiedene Anfragen wurden Zeitmessungen durchgeführt, siehe dazu Tabelle. 4.1.

Relationen	**Stadtwerke Straubing**[1]	**Stadtwerke Passau**[2]	**Alle Autobahnen in Deutschland**[3]
Anzahl Zeilen	2.990	17.196	260.727
Zeit für Parsen	57 - 111 ms	113 - 320 ms	2739 - 2967 ms
Anzahl der OSM-Objekte	888	3.479	60.860
Überprüfungsdauer	12 ms	31 ms	94 ms

[1] (relation["operator"="Stadtwerke Straubing"];node(r)->.nodes;way(r);node(w););out meta;
[2] (relation["operator"="Stadtwerke Passau"]node(r)->.nodes;way(r);node(w););out meta;
[3] (relation["operator"="Bundesrepublik Deutschland"]["network"="BAB"];
node(r)->.nodes;way(r);node(w););out meta;

Tabelle 4.1: Übersicht über Relationen und deren Verarbeitungsdauer in OSMarelmon

Bei näherer Betrachtung der Tabelle fällt auf, dass der Überprüfungsalgorithmus vor allem bei großen Dateien im Vergleich zum Parsen der Dateien kaum Zeit braucht. So benötigt er bei den Dokumenten, die zum Beispiel die Relationen der *Stadtwerke Straubing* oder der *Stadtwerke Passau* enthalten, noch ca. 9,7 - 17,4% bzw. 8,8 - 21,5% der Zeit, die Parsen und Überprüfen zusammen brauchen, bei einem großem Dokument, das *alle Autobahnen in Deutschland* erfasst, jedoch nur 3,1 - 3,3%.
Auch der Speicherverbrauch des Algorithmus hält sich in Grenzen: Durch das sequentielle Laden und Überprüfen der gespeicherten Dokumente befindet sich

durch die *Java Garbage Collection* meist nie mehr als das aktuell zu überprüfende myosm-Objekt in seinen zwei Versionen im Arbeitsspeicher. Im Falle der Datei *Alle Autobahnen in Deutschland*, die als serialisiertes Java Objekt ca. 12 MB groß ist, wird der Arbeitsspeicher laut dem verwendeten Profiling Tool *JProfiler* mit ca. 25 MB belastet - ein absolut verträglicher Wert bei der Größe der heute verwendeten Arbeitsspeicher.

4.3.2 Vergleich mit anderen möglichen Herangehensweisen

Bevor OSMarelmon implementiert wurde, war die große Frage, wie der Abgleich der Relationen effizient stattfinden kann. In diesem Abschnitt werden daher unterschiedliche Herangehensweisen erklärt und erläutert, deren Vor- und Nachteile aufgezeigt und somit die Entscheidung für den Algorithmus begründet.

Verwendung von Augmented Diffs Der implementierte Algorithmus verwendet OSM XML Dateien, die eigentlich nicht darauf spezialisiert sind, Änderungen an OSM-Objekten zu beschreiben. Viel eher dafür geeignet sind die Augmented Diffs, wie in Kapitel 3.2 beschrieben. Augmented Diffs haben den Vorteil, dass sie die Änderungen, die an einem Objekt vorgenommen wurden, bereits enthalten und nicht erst erstellt werden müssen. Weiter könnte man die Überprüfung minütlich durchführen, wobei man sich allerdings gut überlegen müsste, ob dies sinnvoll ist, um Benutzer nicht mit Informationen zu überfluten.
Ein großer Nachteil der Augmented Diffs liegt jedoch darin, dass sie sich nicht auf eine spezielle Region beziehen, sondern auf den ganzen Planet, d.h. in einem Augmented Diff können sich Änderungen in Frankfurt am Main und Änderungen in Madrid befinden. Somit müsste jedes Augmented Diff heruntergeladen und komplett überprüft werden. Der Aufwand, den diese Vorgehensweise nach sich zieht und der geringe Nutzen, vor allem bei wenigen gespeicherten Anfragen in OSMarelmon, stehen in keinem Verhältnis. Der letzte Punkt, der gegen Augmented Diffs spricht, ist der, dass sie immer noch weiter entwickelt werden. Das kann zu höherem Wartungsaufwand dahingehend führen, dass der Algorithmus angepasst werden muss. Dieses Problem kann zwar auch bei der aktuellen Vorgehensweise auftreten, zum Beispiel durch Änderungen an der Overpass API oder am OSM XML Format, da sich vor allem das OSM XML Format schon lange bewährt hat, wird dieser Fall als weniger wahrscheinlich eingeschätzt.

Unstrukturierter Vergleich zwischen den Versionen Eine andere Möglichkeit zwei OSM XML Dateien zu vergleichen und die Unterschiede festzustellen ist, das ganze Dokument von oben nach unten zu durchlaufen ohne dabei Rücksicht auf Zugehörigkeiten zu Relationen oder Wegen zu nehmen. Aufgrund der

vorgegebenen Ordnung von OSM XML Dateien würde das bedeuten, dass zuerst alle Knoten, dann alle Wege und als letztes die Relationen verglichen werden. Diese Vorgehensweise bringt den Vorteil mit sich, dass man sich bei der Überprüfung nicht auf Relationen beschränken muss. Wenn einfach nur objektweise verglichen wird, können grundsätzlich alle Arten von Anfragen überprüft werden, z.B. Anfragen, die Objekte innerhalb einer Bounding Box zurückliefern.
Das Problem bei dieser Lösung ist allerdings der massive Zeitverlust gegenüber dem verwendeten Algorithmus. Durch das unstrukturierte Durchlaufen der Dokumente wird die Zugehörigkeit der OSM-Objekte nicht automatisch mitgeliefert, sondern muss mühsam zusammengesucht werden. Dieses Problem lässt den Algorithmus um ein Vielfaches langsamer laufen, als den verwendeten.

Fazit Nach eingehender Überlegung wurde daher entschieden, einen strukturierten Vergleich von OSM XML Dateien zu implementieren, da diese Vorgehensweise den Kompromiss zwischen Effizienz und Aktualität am besten meistert. Durch diese Lösung kann ein intuitiv verständlicher und einfach wartbarer Algorithmus umgesetzt werden. Zudem sind durch die OSM XML Dateien alle Tags und Metadaten vorhanden, so dass vom Programmierer schnell geändert werden kann, welche Daten an die Nutzer weitergegeben werden sollen und welche nicht.

4.4 Programmfluss

Da nun geklärt wurde, wie Änderungen an den OSM-Daten festgestellt werden können, soll in diesem Abschnitt erläutert werden, wie OSMarelmon auf Benutzereingaben reagiert. Als Beispiel dient dazu das Hinzufügen einer neuen Relation (vgl. dazu auch die Sequenzdiagramme in Abb. 4.6 und Abb. 4.7).

In der Ausgangssituation befindet sich der Benutzer auf der Startseite von OSMarelmon, *start.jsp*. Dort klickt er auf "Add relation to monitor", um neue Relationen überwachen zu lassen. Für den folgenden Abschnitt wird davon ausgegangen, dass der Benutzer korrekte Daten eingegeben hat, die ein nicht-leeres Ergebnis von der Overpass API liefern. Durch den Klick auf den Button wird das Controller-Servlet aufgerufen. Dieses Servlet bearbeitet den Request, indem es den Action-Parameter ausliest. Im Beispiel enthält der Action-Parameter den Wert *add*, so dass das Servlet eine neue *AddRelationAction* erstellt und ausführt. Diese Action holt sich zunächst die Instanz des *FileManager*s, der der Action eine Liste mit den bereits überwachten OSM XML Dateien liefert. Zu diesem Zeitpunkt findet auch die Validierung der vom Benutzer eingegebenen Daten statt. Würde diese fehlschlagen, würde die Action dem Servlet eine Fehlerseite zurückliefern. Im Beispiel sind die Daten allerdings korrekt, so dass ein *AddThread* erstellt werden

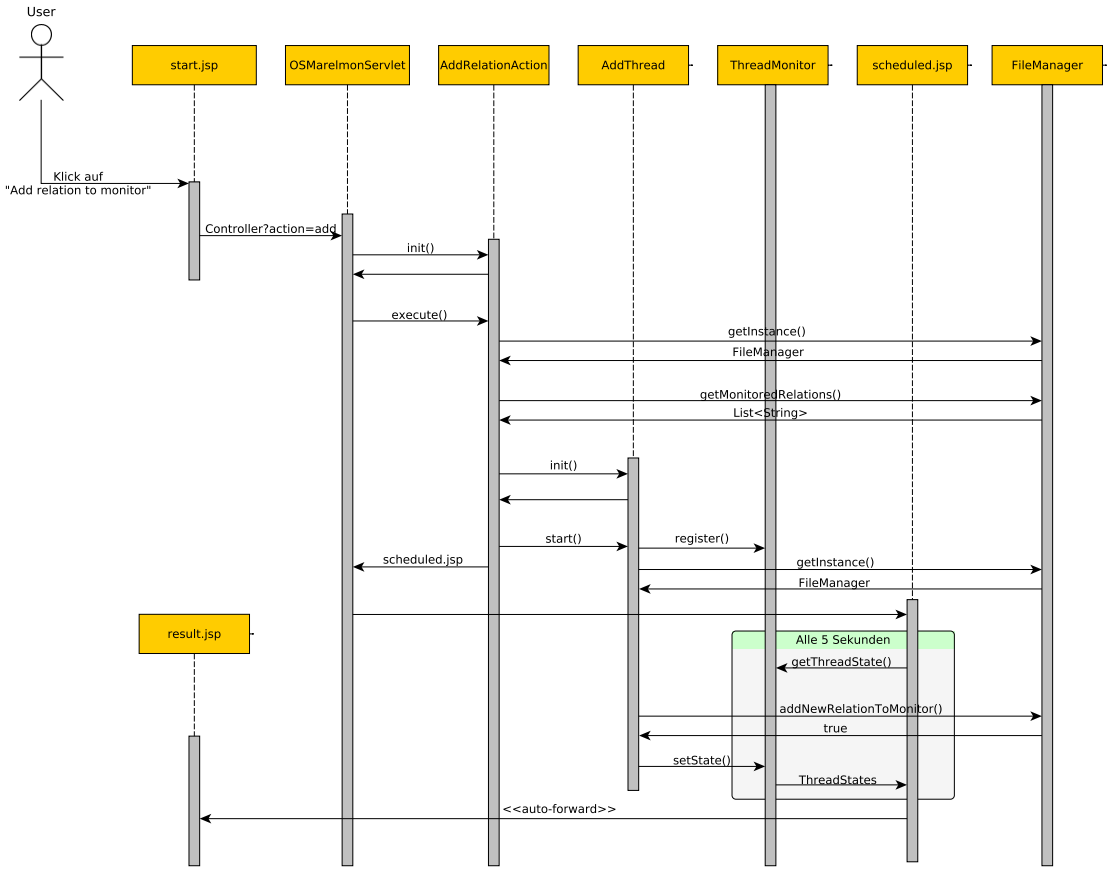

Abbildung 4.6: Sequenzdiagramm für den Ablauf des Hinzufügens einer Relation, Teil I

kann, der die Anfrage von der Overpass API herunterladen und speichern soll. Bevor dies geschieht, registriert sich der Thread beim *ThreadMonitor*. Auf diesen ThreadMonitor kann die JSP-Seite *scheduled.jsp*, die sich alle fünf Sekunden neu lädt, zugreifen, so dass der Benutzer immer über den aktuellen Stand seiner Anfrage informiert werden kann. Wurde die Relation erfolgreich hinzugefügt, wird der Status des Threads auf `SUCCEEDED` gesetzt und beendet. Die Statusänderung wird von der *scheduled.jsp* bemerkt und der Benutzer wird auf die Ergebnisseite, die *result.jsp*, weitergeleitet. Damit ist der Ablauf des Hinzufügens einer Relation beendet.
Da das Speichern und Verwalten der Relation auch einige Funktionsaufrufe mit sich bringt, wurde dieser Ablauf in eigenes Sequenzdiagramm ausgelagert (Abb. 4.7): Nachdem der AddThread beim FileManager das Hinzufügen angestoßen hat, wird zuerst aus den vom Benutzer eingegebenen key-value Paaren eine Overpass

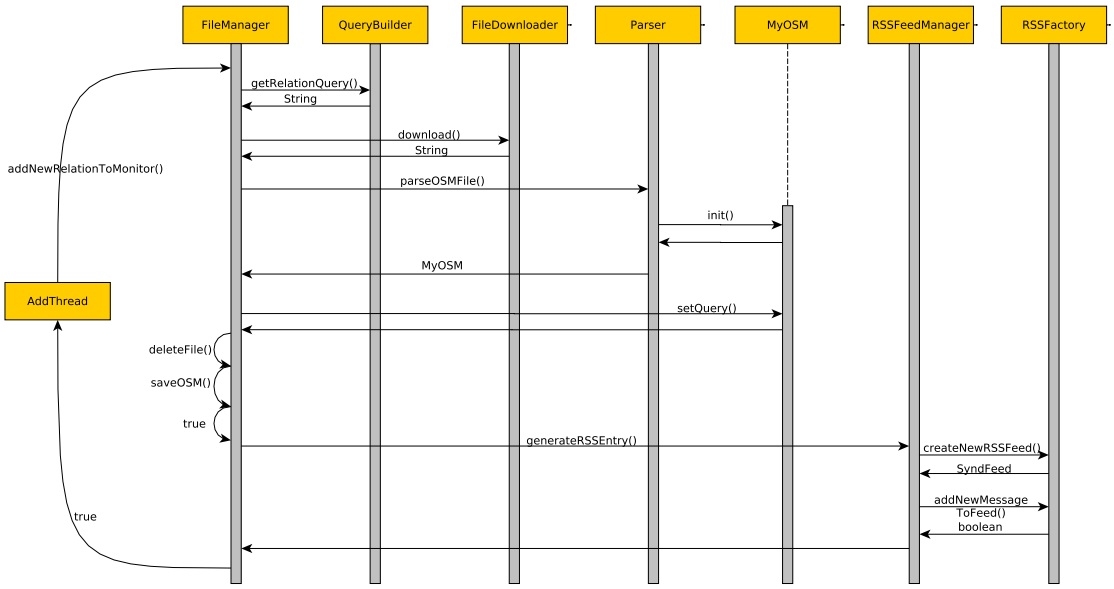

Abbildung 4.7: Sequenzdiagramm für den Ablauf des Hinzufügens einer Relation, Teil II

Query generiert. Danach werden mittels dieser Query die entsprechenden OSM-Daten von der Overpass API geladen und temporär auf der Festplatte gespeichert. Anschließend wird diese Datei geparst und mit dem Aufruf `deleteFile()` gelöscht, um nicht unnötig Festplattenspeicher zu verbrauchen. Dem aus dem Parsen resultierenden myosm-Objekt wird aus verwaltungstechnischen Gründen die Query hinzugefügt und danach gespeichert. Um den Benutzern einen RSS Feed zur Verfügung zu stellen, wird über den *RSSFeedManager* und eine *RSSFactory* ein neuer Feed erstellt und eingerichtet.

4.5 Vorstellung und Vergleich vorhandener Programme

In diesem Abschnitt werden Programme vorgestellt, die eine zu OSMarelmon ähnliche Funktionalität haben. Dabei werden auch die Unterschiede und Gemeinsamkeiten aufgezeigt.

4.5.1 OSM History Viewer

Das erste Tool ist der *OSM History Viewer* (OSMHV)[1]. Mit dem History Viewer können hauptsächlich Änderungen, die an OSM-Daten vorgenommen wurden, sichtbar gemacht werden. Im Webinterface stehen dem Nutzer zwei Möglichkeiten offen, Änderungen einzusehen: Zum einen kann der Benutzer eine Changeset-ID angeben, zum anderen können per ID die Änderungen an Relationen angezeigt werden. Wird ein Changeset mit OSMHV gesucht und analysiert, so werden die bearbeiteten Daten wie folgt angezeigt [15]: Alle Wege, die vom angegebenen Changeset betroffen sind, werden in Segmente unterteilt, wobei ein Segment die Verbindung zwischen zwei Knoten darstellt. Jedes Segment wird nun dahingehend untersucht, ob es bewegt, erstellt oder gelöscht wurde. Falls ein Segment oder seine Position lediglich vor der Änderung vorhanden war, wird es rot markiert. War es nur nach dem Änderung vorhanden wird es grün eingefärbt. Segmente, die sich nicht verändert haben, werden blau markiert (siehe dazu auch Abb. 4.8).

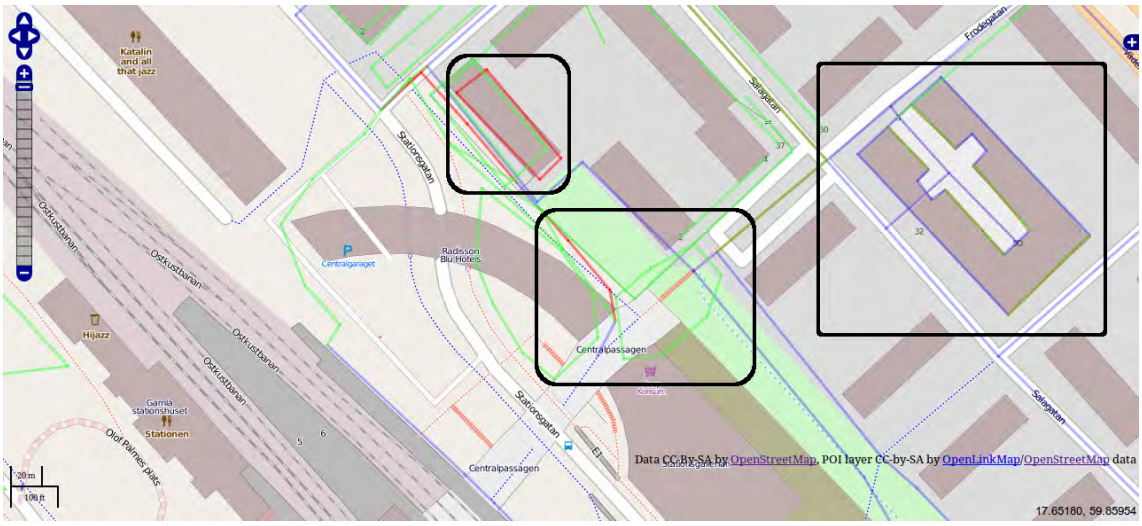

Abbildung 4.8: Beispielhafte Visualisierung eines Changesets mit OSMHV[2]

Die zweite Möglichkeit Änderungen einzusehen, ist über die Relationenabfrage. Der Nutzer muss wieder eine ID eingeben (dieses Mal natürlich die ID einer Relation) und wird dann zu einer Seite weitergeleitet, die ihm alle Changesets auflistet, die etwas an der Relation verändert haben. Den Changesets werden verschiedene Farben zugewiesen und auf der rechten Seite wird die Relation mit den unterschiedlichen Farben angezeigt (siehe Abb. 4.9).

[1] Homepage: `http://osmhv.openstreetmap.de/index.jsp`
[2] ©OpenStreetMap.org contributors, Data: ODbL, Map: cc-by-sa

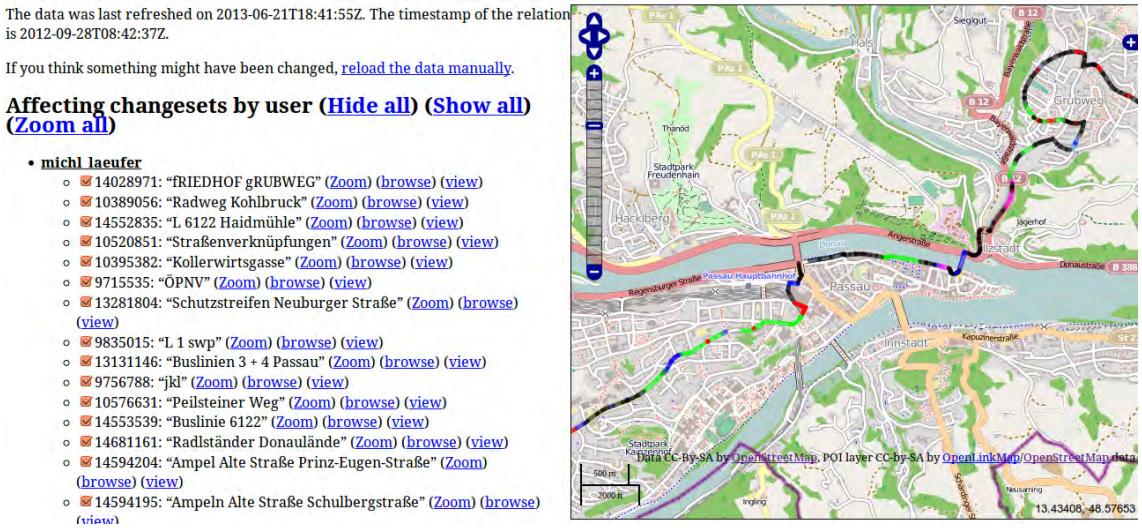

Abbildung 4.9: Screenshot von OSMHV - Visualisierung von Änderungen an einer Relation

Die Vorzüge des OSMHV liegen auf der Hand: Die Visualisierung der Änderungen in der OSM-Karte ist ein sehr schönes Feature, genauso wie das Betrachten der Changesets. Will man OSMHV jedoch nutzen, um Relationen zu überwachen, ist dieses Programm keine gute Wahl: Zum einen kann der Benutzer Relationen nicht über Tags suchen, sondern muss die ID's kennen und zum anderen wird er nicht automatisch über Änderungen informiert, wenn eine Änderung stattgefunden hat. Die Suche per ID bringt auch den Nachteil mit sich, dass nicht mehr als eine Relation auf einmal betrachtet werden kann.

4.5.2 OSM-Watch

Das zweite hier vorgestellte Programm heißt *OSM Watch* (Homepage: `http://osm102.openstreetmap.fr/~zorglub/watch/`), wobei diese Applikation während der Zeit, in der diese Bachelorarbeit entstanden ist, nicht erreichbar war, weshalb alle Informationen darüber von der entsprechenden OSM-Wiki Seite stammen [16]. Mit OSM Watch können folgende Elemente beobachtet bzw. überwacht werden:

- eine Bounding-Box
- eine Relation
- ein Tag (innerhalb einer Bounding-Box)
- ein Benutzer.

Die Benachrichtigung kann wie bei OSMarelmon über einen RSS-Feed erfolgen. Dieses Programm ist sehr ähnlich zu OSMarelmon, es wird sogar einiges an Zusatzfunktionalität angeboten. Laut Wiki-Seite erfolgen die Benachrichtigung der Abonnenten der RSS Feeds in Echtzeit. Wie bereits gesagt, konnte das Programm nicht getestet werden, da die Website nicht erreichbar war.

4.5.3 OSM Watch List

Das letzte und vielleicht interessanteste hier vorgestellte Programm ist *OSM Watch List* (OWL)[3]. OWL ist ein Dienst, der Änderungen an den OSM-Daten überwacht, verarbeitet und speichert. Ein Haupt-Feature von OWL ist, dass Änderungen nicht mit entsprechenden Bounding-Boxen überwacht werden, sondern mit kleinen Tiles[4], die besser zur Visualisierung geeignet sind. So ist es möglich, nur die relevanten Änderungen anzuzeigen und nicht alle Änderungen mit überlappenden Bounding-Boxen [20]. Wie in Abbildung 4.10 gezeigt, zoomt man auf der OSM-Karte auf eine passende Höhe, bis erste Änderungen am linken Rand angezeigt werden. Gleichzeitig erscheinen in der Karte blaue Punkte, die die Änderungen in der Karte markieren. Klickt man dann auf der linken Seite eine Änderung an, öffnet sich ein Untermenü, das die Tags des Objekts anzeigt und die Änderungen, die dort vorgenommen wurden, markiert.

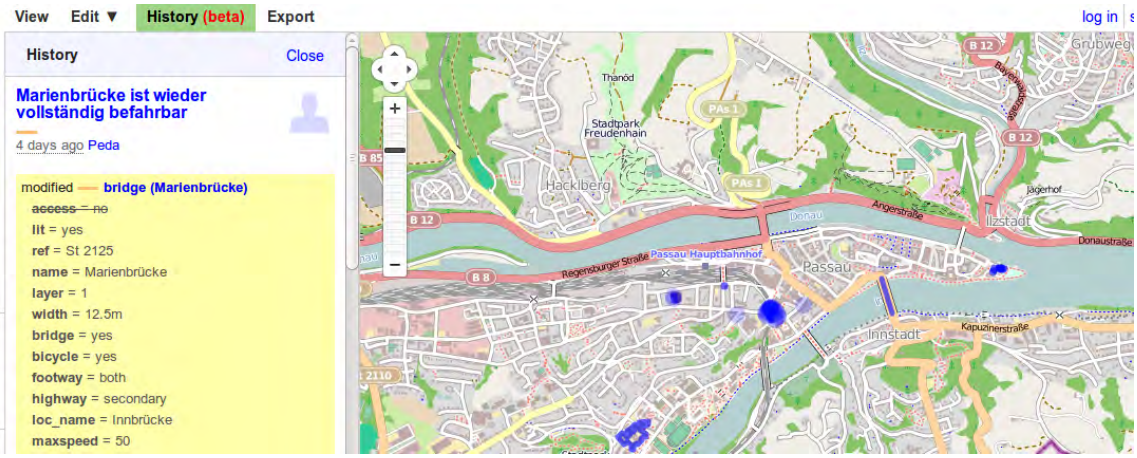

Abbildung 4.10: Screenshot von OWL - Ansicht von Passau

Bei diesem Dienst ist die einfache Bedienung und die sehr ansehnliche Visualisierung hervorzuheben. Zudem besteht die Möglichkeit, einen RSS Feed für die

[3]Homepage: `http://owl.openstreetmap.org/`

[4]Ein Tile ist ein rechteckiger Ausschnitt auf der Karte, der ein bestimmtes Gebiet abdeckt. Bei geringer Zoomstufe decken die Tiles ein großes Gebiet ab, zeigen dafür aber wenig Details an.

betrachteten Tiles zu abonnieren. Es wird vermutet, dass dieser Feed den Abonnenten informiert, sobald ein neues Changeset erstellt wird, das die beobachteten Tiles betrifft. Alles in allem ist OWL ein sehr interessantes Tool, das sich allerdings im Moment noch in der Betaphase befindet.

4.6 Erweiterungsmöglichkeiten

Dieser Abschnitt zeigt auf, welche Erweiterungen für OSMarelmon interessant sein könnten. Manche der vorgeschlagenen Erweiterungen wurden bereits umgesetzt, sind aber für eine Webanwendung von eher geringem Nutzen.

4.6.1 Erweiterte Überwachungsmöglichkeiten

Da OSMarelmon nur Relationen überwachen kann, ein Benutzer allerdings vielleicht eine Stadt oder einen Teil davon überwachen möchte, wäre eine nützliche Erweiterung, die Überwachung von Bounding-Boxen einzubauen. Um dieses Feature umzusetzen, müsste dann allerdings der in Kapitel 4.2 besprochene Algorithmus zum unstrukturierten Vergleich von OSM XML Dateien implementiert werden, da bei einer Abfrage der Daten innerhalb einer Bounding-Box keine Aussagen über deren Inhalt getroffen werden kann. Sollte dieses Feature in der Webanwendung umgesetzt werden, muss zudem beachtet werden, dass die Bounding-Boxen sehr nutzerspezifisch sind, d.h. der eine Nutzer versteht unter der Bounding-Box "Passau"den geographischen Bereich, der von der Bounding-Box mit den geographischen Punkten 13.42697, 48.55934, 13.46005, 48.57686 (maximales Breiten/Längengrad, minimales Breiten/Längengrad) eingegrenzt wird und ein anderer Nutzer würde gerne aber ein bisschen mehr überwachen und registriert deshalb die Bounding-Box 13.3932, 48.5505, 13.4891, 48.5855. Auf diese Weise können große Redundanzen in der Datenhaltung entstehen.

4.6.2 Freie Overpass Queries

Ähnlich zu den erweiterten Überwachungsmöglichkeiten ist die Idee, den Nutzer selbst die Overpass Queries gestalten zu lassen. Da OSMarelmon nur Relationen überwachen kann, werden auch nur Relationen-Anfragen nach dem key-value Schema zugelassen. Um dem Benutzer mehr Freiheiten zu geben, könnte man ihm also gestatten, eigene Queries zu definieren. Dies setzt natürlich die Kenntnis der Overpass QL voraus. Da auch beim Ergebnis von freien Overpass Query keine Aussagen über die Struktur der Datei gemacht werden können, muss auch hierfür der Algorithmus zum unstrukturierten Vergleich implementiert werden.

4.6.3 GUI Applikation

Ein Feature, das bereits umgesetzt wurde, aber nicht verwendet wird, ist die Implementierung von OSMarelmon als normale Java-Anwendung auf dem eigenen Rechner. Die Idee hierbei ist, dass sich jeder Nutzer das Programm auf seinem Rechner installieren kann und Überprüfungen nur stattfinden, wenn die Anwendung läuft. Die Meldungen über Änderungen könnten dann über lokale RSS-Dateien ablaufen oder in einem extra Fenster betrachtet werden. In Abbildung 4.11 wird der Prototyp von OSMarelmon als Java Applikation gezeigt.

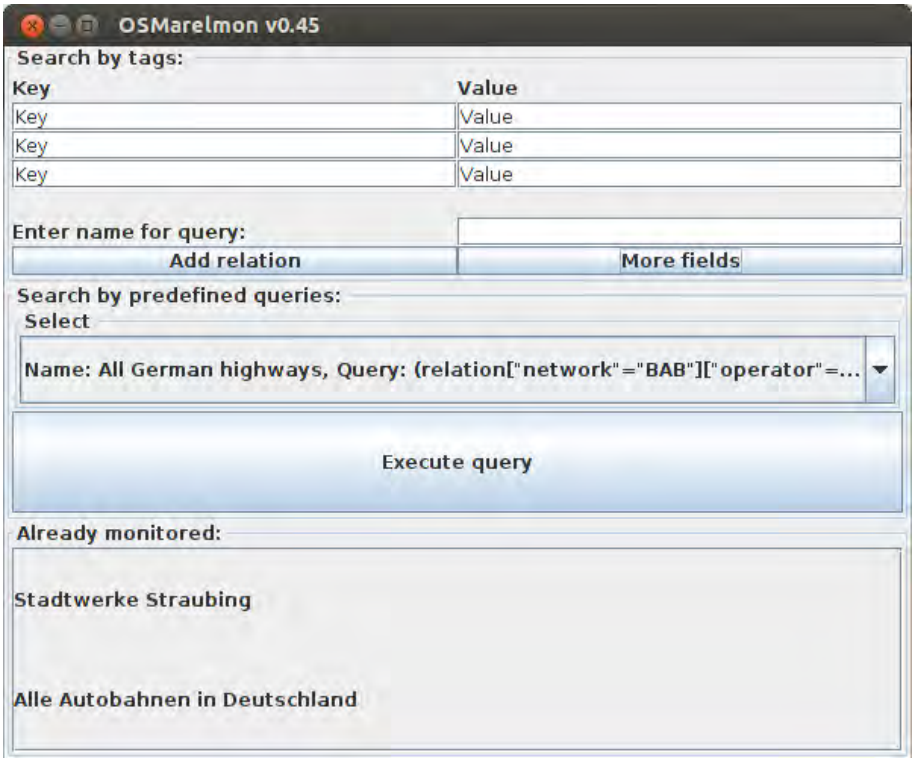

Abbildung 4.11: Screenshot von OSMarelmon als Java Applikation

Grundsätzlich ähnelt der Aufbau dem der Webanwendung: Am oberen Rand befindet sich die Eingabe der key-value Paare zum Definieren einer neuen Anfrage und am unteren Rand werden die bereits überwachten Relationen angezeigt.
Für eine solche Anwendung, die von nur einem Benutzer genutzt wird, kann es auch nützlich sein, vordefinierte Queries anzubieten. Im Screenshot ist zum Beispiel in der Mitte die vordefinierte Query "All German highways, ..." ausgewählt. Um solch ein Feature jedoch brauchbar zu gestalten, sollte eine große Zahl von vordefinierten Anfragen angeboten werden.

Kapitel 5

Nutzen für das OpenStreetMap Projekt

Das Programm wurde noch während der Fertigstellung dieser Arbeit für die OpenStreetMap Community zugänglich gemacht, um es zum einen testen zu können und zum anderen ein Feedback der Community bezüglich der Nützlichkeit und der allgemeinen Bedienungsfreundlichkeit zu erhalten. In diesem abschließenden Kapitel wird auf die Rückmeldungen der Community eingegangen und zu einem Fazit zusammengefasst.
Das Programm wurde zwei Wochen lang rege benutzt: Es wurden 26 Anfragen der Community gestellt, die insgesamt 152 MB OSM-Rohdaten umfassten, was geschätzt ca. 2.6 Millionen Zeilen Daten entspricht. Diese Datenmenge wurde innerhalb von 4 Minuten von der Overpass API heruntergeladen und auf Änderungen überprüft.
Von den Nutzern gab es meist positives Feedback: Bezüglich der Nützlichkeit des Programms gingen durchweg positive Rückmeldungen ein. Die meisten Benutzer begrüßten den Dienst, da etwas Vergleichbares noch nicht existiert. Besonders der Aspekt, dass auch Benachrichtigungen stattfinden, wenn sich die geographische Position in Knoten verändert, wurde als sehr sinnvoll betrachtet. Die Nutzer zeigten sich sehr interessiert am Programm, was besonders am Anfang durch viele Fragen deutlich wurde, als sich in den überwachten Relationen noch nicht viel getan hatte und dementsprechend wenig Änderungen über die Feeds verteilt wurden ("merkt das Programm eigentlich, wenn..." oder "kann das Programm...").
Bezüglich der Lesbarkeit und der Überschriften der Feeds gab es einige Verbesserungsvorschläge, wie zum Beispiel die Anzahl der Änderungen in die Überschrift mitzunehmen oder die Overpass Query zu verlinken.
Auch an der Website gab es ein paar Wünsche, wie man das Programm noch attraktiver machen könnte. Zum einen wurde vorgeschlagen, ein Eingabefeld einzubauen, über das die Nutzer Relationen direkt per ID eingeben können, um nicht

die key-value Paare benutzen zu müssen, zum anderen, dass man das Ergebnis seiner Anfrage, also die Relationen, die von den key-value Paaren und der Overpass Query "getroffen" werden, ansehen kann.
Alles in allem war das Feedback äußerst hilfreich, um die Nützlichkeit und die Bedienungsfreundlichkeit des Programms einzuschätzen. Um den Service weiter erhalten und Verbesserungswünsche der Community schneller umsetzen zu können, wird der Quellcode nach Abgabe dieser Arbeit über ein git-Repository öffentlich zugänglich gemacht werden.

Literaturverzeichnis

[1] C. Universität Passau Bachmaier. *Programmierung II - Objektorientiertes Programmieren in Java*. 2011.

[2] H. Universität Passau Kosch. *Web Engineering*. 2013.

[3] Pentagon. Openstreetmap relation: pentagon (89605). `http://www.openstreetmap.org/browse/relation/89605`, 2013. [Online; accessed 08-June-2013].

[4] F. Ramm and J. Topf. *OpenStreetMap: Die freie Weltkarte nutzen und mitgestalten*. Lehmanns Media GmbH, 2010.

[5] ROME. Rome – rome tools. `https://rometools.jira.com/wiki/display/ROME/Home`, 2013. [Online; accessed 11-June-2013].

[6] OSMDE009 OSM Talk. Die overpass api. `http://blog.openstreetmap.de/2012/12/osmde009-osm-talk-die-overpass-api/`, 2012. [Online; accessed 03-July-2013].

[7] w3c. w3c – rss tutorial. `http://www.w3schools.com/rss/`, 2013. [Online; accessed 11-June-2013].

[8] OpenStreetMap Wiki. Osmchange — openstreetmap wiki,. `http://wiki.openstreetmap.org/w/index.php?title=OsmChange&oldid=841315`, 2012. [Online; accessed 24-June-2013].

[9] OpenStreetMap Wiki. De:key:crossing — openstreetmap wiki,. `http://wiki.openstreetmap.org/w/index.php?title=DE:Key:crossing&oldid=909148`, 2013. [Online; accessed 8-June-2013].

[10] OpenStreetMap Wiki. De:planet.osm — openstreetmap wiki,. `http://wiki.openstreetmap.org/w/index.php?title=DE:Planet.osm&oldid=869403`, 2013. [Online; accessed 8-June-2013].

[11] OpenStreetMap Wiki. History of openstreetmap — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=History_of_OpenStreetMap&oldid=897370, 2013. [Online; accessed 9-July-2013].

[12] OpenStreetMap Wiki. Josm — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=JOSM&oldid=912922, 2013. [Online; accessed 26-June-2013].

[13] OpenStreetMap Wiki. Josm/search function — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=JOSM/Search_function&oldid=896744, 2013. [Online; accessed 26-June-2013].

[14] OpenStreetMap Wiki. Main page — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Main_Page&oldid=888363, 2013. [Online; accessed 27-May-2013].

[15] OpenStreetMap Wiki. Osm history viewer — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=OSM_History_Viewer&oldid=896253, 2013. [Online; accessed 21-June-2013].

[16] OpenStreetMap Wiki. Osm-watch — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Osm-watch&oldid=858935, 2013. [Online; accessed 21-June-2013].

[17] OpenStreetMap Wiki. Osm xml — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=OSM_XML&oldid=907375, 2013. [Online; accessed 8-June-2013].

[18] OpenStreetMap Wiki. Overpass api — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Overpass_API&oldid=905034, 2013. [Online; accessed 10-June-2013].

[19] OpenStreetMap Wiki. Overpass api/augmented diffs — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Overpass_API/Augmented_Diffs&oldid=905050, 2013. [Online; accessed 19-June-2013].

[20] OpenStreetMap Wiki. Owl — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=OWL&oldid=874805, 2013. [Online; accessed 21-June-2013].

[21] OpenStreetMap Wiki. Stats — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Stats&oldid=904323, 2013. [Online; accessed 27-May-2013].

[22] OpenStreetMap Wiki. Tags — openstreetmap wiki,. http://wiki.openstreetmap.org/w/index.php?title=Tags&oldid=867416, 2013. [Online; accessed 9-August-2013].