

# OPENSTREETMAP-IN-A-BOX

## GESAMTDOKUMENTATION

**Gruppenmitglieder:** *Fabio Renggli*  
*Michael Huber*  
*Roland Hof*

**Document released:** *11. June 2009*

## ABSTRACT DEUTSCHE VERSION

### BESCHREIBUNG

OpenStreetMap (OSM) ist das 'Wikipedia' der Landkarten. Es ist ein Software-Projekt mit dem Ziel, eine frei nutzbare Weltkarte zu erstellen. OSM wächst seit ca. 2007 exponentiell.

Im Gegensatz anderen Kartendiensten wie zum Beispiel Google Maps ist OSM detaillierter, aktueller und vor allem: Die gesammelten Daten können frei genutzt werden, unter anderem in Navigationssystemen (GPS) und Smartphones. Im Rahmen dieser Bachelorarbeit wurde vor allem ein eigenständiger OSM-Konverter entwickelt, welcher die bestehenden OSM-Daten in eine interne Datenstruktur importiert. Die so aufbereiteten OSM-Daten werden über die Webservices eines professionellen Geographischen Informationssystems (GIS) zur Verfügung gestellt. Beispielsweise für Firmen-Websites oder Hilfsprojekte.

### ZIELE

Für die Bachelorarbeit wurden folgende Ziele vereinbart:

- Die von OSM zur Verfügung gestellten Daten sollen auf einen eigenständigen Server importiert werden.
- Die Daten auf dem Server sollen mittels einer Aktualisierungsprozedur aktuell gehalten werden.
- Verwendung von GeoServer, um mit Hilfe der Webservices die Daten für externe Betrachter verfügbar zu machen.
- Eine Website entwerfen, welche mit Hilfe von OpenLayers die Daten für demonstrative Zwecke visualisiert.

### LÖSUNG

Die von OSM aufbereiteten Export-Dateien (sog. ‚Planet Files‘) werden mithilfe einer Java-Applikation konvertiert und in eine PostgreSQL Datenbank importiert. Zur effizienten Verwaltung der Geometriedaten wird die PostGIS-Erweiterung eingesetzt. Die vom OSM-Projekt generierten Differenzial-Files werden in definierbaren Zeitintervallen (z.B. jede Stunde) zu den bestehenden Daten hinzugefügt. Da die Datenmenge sehr gross ist und stetig anwächst, ist es auch möglich nur Teilbereiche der Daten zu importieren.

Die Datenbankstruktur ist so angelegt, dass der GeoServer daraus vorberechnete Kacheln (sogenannte Tiles) generieren kann, welche eine verbesserte Performanz bei der Ansicht ermöglichen. Auf die Daten kann über den Web Map Service (WMS) oder den Web Feature Service (WFS) zugegriffen werden. Um die Performance weiter zu verbessern wird auf dem Server GeoWebCache eingesetzt. Dieser kann die vorhandenen Tiles direkt ausliefert und greift nur auf die WMS-Schnittstelle des Geoservers zurück, wenn ein Tile neu generiert werden muss.

## ABSTRACT ENGLISH VERSION

### DESCRIPTION

The open source project OpenStreetMap (OSM) is the ‚Wikipedia‘ of maps. OSM is a software project with the goal to build freely usable maps of the world. It is growing exponentially since ca. 2007. In contrast to other map services like Google Maps, OSM is more detailed, more often updated and especially: Among others, the collected data is free to use in other applications such as navigation systems (GPS) and smart phones.

During this bachelor thesis, an independent converter was build, which parses the existing OSM data and imports it into an internal datastructure. The so preprocessed OSM data can be accessed through webservices of a professional geographical information system (GIS), for example to use it on a company website or in a relief project.

### GOALS

- Importing of the provided data from OSM on an independent server.
- The data on the server should be held up-to-date through a scheduling procedure.
- Usage of GeoServer to provide webservices with the data for external access.
- Designing a website, which visualizes the provided tiles using OpenLayers.

### SOLUTION

The OSM export files (so called ‚Planet Files‘) are converted and imported into a PostgreSQL database using a java application called osm2gis. The PostGIS add on is used for the efficient management of the geometry data.

In well-defined time intervals (e.g. every hour), the differential files generated by OSM are imported into the present database. Given that the amount of data is very big and is growing steadily, it is possible to import only part of the data.

A data structure is applied, that the GeoServer can render tiles, which improves the performance while viewing the maps. The data can be accessed through the Web Map Service (WMS) and the Web Feature Service (WFS).

To improve the performance even more, GeoServer uses GeoWebCache to cache and to deliver the rendered tiles directly. It only accesses the WMS interface of GeoServer, when a tile does not yet exist or has to be re-rendered.

## 1 MANAGEMENT SUMMARY

### 1.1 PROBLEM DEFINITION

Most cartographic visualization requires a base map layer to provide geographic context. Current base map products are either rather expensive and sometimes outdated or not available for certain regions (i.e. cadastral data or Google Maps). Up-to-date base maps become more and more important when setting up a spatial infrastructure for companies, non-governmental organizations or the public sector.

OpenStreetMap (OSM) is a crowd sourcing project which aims to provide free geographic data (under a viral GPL-like license). OSM itself is run by many open source tools with a zoo of languages written by volunteers. Thus, in order to setup an OSM server there is quite a heterogeneous bunch of software involved with obscure dependencies which is only partially documented. In addition the OSM community doesn't care much about international geographic information standards like those from Open Geospatial Consortium (OGC). And for some projects it's important to have an own map server either because it is sometimes offline or because it needs to be reliable and fast.

A major goal of this project was to create an easy to use installer which installs OSM out of the box as a dedicated map server including well-known OGC web services which keeps data in sync with the original OSM data. This includes several parts as follows:

- **Part I: Importer**  
Development of an importer which can import OSM files into a spatial database and hold this database up to date.
- **Part II: Geo-information server**  
Configuration and installation of a geo-information server called GeoServer. Including the configuration of a Web Map Service (bitmap), a Web Feature Service (vector data, read-only) and a Web Map Tiling Service (tiled raster data).
- **Part III: Website**  
Creation of a website to demonstrate the project.
- **Part IV: Write access -optional-**  
The write access to the create database is possible and the changes made local can be synchronized with OSM API.

### SIMILAR PROJECTS

A lot of projects for data import and export already exist under the roof of OpenStreetMap. But none of them fulfills all the requirements mentioned above. For illustration purposes we like to mention two prevalent tools.

Tool	Description	Weblink
<b>osmosis</b>	Most powerful tool for OSM. Mostly used to import / export data in OSM format	<a href="http://wiki.openstreetmap.org/wiki/Osmosis">http://wiki.openstreetmap.org/wiki/Osmosis</a>
<b>osm2pgsql</b>	osm2pgsql is a utility program that converts OpenStreetMap (OSM) data into a PostgreSQL format.	<a href="http://wiki.openstreetmap.org/wiki/Osm2pgsql">http://wiki.openstreetmap.org/wiki/Osm2pgsql</a>

## 1.2 APPROACH

The start of the project was mostly used for technical studies on OpenStreetMap, the map server (GeoServer) and the database PostgreSQL with PostGIS, the additional add-in for spatial data. With the big picture created, the work on the project began with the importer and parallel work on the configuration of the map server. These tasks claimed almost the whole project time and at the end a website was created to present the project. The optional part IV was not implemented due to its complexity and time shortage.

The OpenStreetMap-in-a-Box team consists of Michael Huber, Fabio Renggli and Roland Hof.

The progress of the project was controlled and emerging question were discussed on a weekly meeting with supervising professor Prof. Stefan Keller.

## 1.3 RESULTS

The result of this project is a working 'easy-to-use' installer which installs the following, self developed or configured components:

- osm2gis with initial import and differential update for OpenStreetMap data into a PostGIS database
- Preconfigured GeoServer including the style configuration (SLD) to render tiles.
- Website with OpenLayers to display the generated tiles.
- Usage of GeoWebCache to boost the performance on the website.

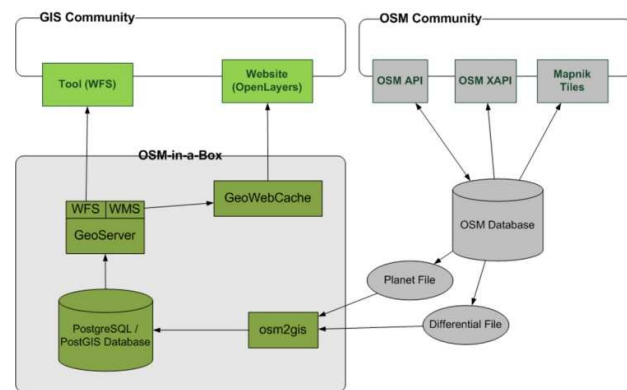


Abbildung 1: Ausgangslage (rechts) mit neuen Erweiterungen (links)

## GOAL ATTAINMENT

All primary goals have been achieved. Some subordinate tasks have been added and fulfilled like the possibility to use not only the built-in scheduler but also an external one or the feature that a scheduled update which crashed can catch-up to be up to date without any further user interaction. The synchronization back to the OSM-API have not been achieved due to time shortage.

### 1.3.1 EXTERNAL RESSOURCES

For this project we used several external libraries such as the JDBC driver for PostgreSQL or a java scheduler called Quartz which used several other common libraries from Apache. For the creation of the SQL statements GeoTools has been used to compare polygons with each other. The read-part of the xml file has been done with bzip2 library which is able to read a zipped file as stream. Argument parsing is done with JASP.

For testing and logging the application uses junit and log4j.

## 1.4 FUTURE WORK

As seen in the results chapter above, we achieved the primary goals of this project. But of course future work on this project can still be done. From our point of view, we see the following tasks that could be done in a future project:

- Addition of more OSM tags to our database by adding new entries to mapping.xml
- Improvement of style configuration (SLD) for better and more detailed map tiles.
  - Could also be done with database views. Of course this would also mean improvements on the SLDs.
- Writing
  - Write via WFS and GeoServer into our database.
  - Find the new entries and add them over the OSM API to their database.

More Information on: <http://dev.ifs.hsr.ch/osminabox>

## 2 INHALTSVERZEICHNIS

<b>ABSTRACT DEUTSCHE VERSION .....</b>	<b>2</b>
BESCHREIBUNG .....	2
ZIELE.....	2
LÖSUNG .....	2
<b>ABSTRACT ENGLISH VERSION .....</b>	<b>3</b>
DESCRIPTION .....	3
GOALS .....	3
SOLUTION .....	3
<b>1 MANAGEMENT SUMMARY .....</b>	<b>4</b>
1.1 PROBLEM DEFINITION .....	4
1.2 APPROACH.....	5
1.3 RESULTS .....	5
1.3.1 External ressources.....	5
1.4 FUTURE WORK .....	6
<b>2 INHALTSVERZEICHNIS .....</b>	<b>7</b>
<b>3 AUFGABENSTELLUNG .....</b>	<b>10</b>
3.1.1 Einführung.....	10
3.1.2 Aufgabenstellung.....	10
3.1.3 Hinweise .....	11
3.1.4 Randbedingungen, Infrastruktur, Termine und Beurteilung.....	11
<b>4 TEIL I TECHNISCHER BERICHT .....</b>	<b>12</b>
4.1 PROBLEMBSTELLUNG.....	12
4.2 ZIEL DER ARBEIT .....	13
4.3 DIE WICHTIGSTEN ANFORDERUNGEN .....	14
4.4 ERGEBNISSE.....	15
4.4.1 Einbettung des OSM-in-a-BOX servers .....	15
4.4.2 osm2gis initialer import.....	16

4.4.3	<i>osm2gis update service</i> .....	16
4.4.4	<i>osm2gis Architektur</i> .....	17
4.5	SCHLUSSFOLGERUNGEN .....	18
4.6	AUSBLICK .....	18
<b>5</b>	<b>TEIL II SW-PROJEKTDOKUMENTATION</b> .....	<b>19</b>
5.1	ANFORDERUNGSSPEZIFIKATION .....	19
5.1.1	<i>Anforderung an die Arbeit</i> .....	19
5.1.2	<i>Funktionale Anforderungen</i> .....	21
5.1.3	<i>Nicht-funktionale Anforderungen</i> .....	22
5.1.4	<i>Use Cases</i> .....	23
5.1.5	<i>WMS abfrage</i> .....	24
5.1.6	<i>Sequenzdiagramm</i> .....	26
<b>5.2</b>	<b>DOMAINMODELL</b> .....	<b>27</b>
5.2.1	<i>Modell</i> .....	27
5.2.2	<i>Konzeptbeschreibungen</i> .....	28
<b>5.3</b>	<b>DESIGN</b> .....	<b>29</b>
5.3.1	<i>Physical Architecture</i> .....	29
5.3.2	<i>Logical Architecture</i> .....	30
5.3.3	<i>General functionality</i> .....	32
5.3.4	<i>Application initialization</i> .....	32
5.3.5	<i>Introducing the application context</i> .....	34
5.3.6	<i>Importer: parsing process</i> .....	35
5.3.7	<i>Update scheduler</i> .....	43
5.3.8	<i>Database-Layer: Data Migration</i> .....	45
5.3.9	<i>Differential Update</i> .....	48
5.3.10	<i>Node Handling</i> .....	50
5.3.11	<i>Consistency Service</i> .....	65
5.3.12	<i>Data Mapping</i> .....	66
5.3.13	<i>Database</i> .....	69



5.3.14	<i>How To's</i> .....	70
5.4	PROJEKTMANAGEMENT .....	71
5.4.1	<i>Projektorganisation</i> .....	71
5.4.2	<i>Managementabläufe</i> .....	71
5.4.3	<i>Risikomanagement</i> .....	75
5.4.4	<i>Arbeitspakete</i> .....	76
5.4.5	<i>Infrastruktur</i> .....	82
5.4.6	<i>Qualitätssicherung</i> .....	82
5.5	PROJEKTMONITORING .....	83
5.5.1	<i>Soll-Ist-Zeit-Vergleich</i> .....	83
5.5.2	<i>Codestatistik</i> .....	84
5.5.3	<i>Sitzungsprotokolle</i> .....	84
5.6	SOFTWAREDOKUMENTATION .....	85
5.6.1	<i>Server Konfiguration</i> .....	85
5.6.2	<i>Installationsmanual</i> .....	92
<b>6</b>	<b>ANHANG</b> .....	<b>97</b>
6.1	ERFAHRUNGSBERICHTE .....	97
6.1.1	<i>Michael Huber</i> .....	97
6.1.2	<i>Roland Hof</i> .....	98
6.1.3	<i>Fabio Renggli</i> .....	99
6.2	INHALT DER CD .....	100
6.3	ANHANG B GLOSSAR UND ABKÜRZUNGSVERZEICHNIS .....	100
6.4	ANHANG C LITERATUR- UND QUELLENVERZEICHNIS .....	101
6.5	ANHANG D EIGENHÄNDIGKEITSERKLÄRUNG .....	102
6.5.1	<i>Michael Huber</i> .....	102
6.5.2	<i>Roland Hof</i> .....	102
6.5.3	<i>Fabio Renggli</i> .....	102

## 3 AUFGABENSTELLUNG

**Autoren/Studenten:**

Michael Huber, Fabio Renggli und Roland Hof, HSR,  
Abteilung Informatik

**Verantwortlicher/  
Betreuer:**

Prof. Stefan Keller, HSR, Abt. Informatik

**Partner (Firma oder Verwaltung),  
externer Betreuer:**

(Open Source Community)

### 3.1.1 EINFÜHRUNG

OpenStreetMap (OSM) ist das Wikipedia der Landkarten. Es ist ein Software-Projekt mit dem Ziel, eine für jeden frei nutzbare Weltkarte zu erstellen. OSM wächst seit 2007 ähnlich rasant wie Wikipedia vor fünf Jahren. Im Gegensatz zu Google Maps ist OSM zum Teil detaillierter und aktueller und vor allem kann es frei genutzt werden, z.B. in eigenen Webseiten, Navigationssystemen (GPS) und Mobiles.

Hinter OSM bestehen zurzeit v.a. folgende Webdienste: Der Tile-Server (à la Google Maps API) und das OSM API (read/write). Dazu kommen sog. Renderer, die mit Hilfe von Konfigurationsdateien („Styles“) aus den Geometriedaten die Kartengrafik erzeugen und dann zu ‚Kacheln‘ (engl. Tiles) aufbereiten. Diese Softwarekomponenten sind zurzeit in verschiedenen Sprachen geschrieben und schwer installierbar. Zudem fehlen international verbreitete Geo-Webdienste-Standards, wie z.B. der Web Map Service (WMS). Auf der anderen Seite gibt es einen ‚The Open Geo-Stack‘, der aus bewährter Open Source-Software besteht, wie sie auch an der HSR (Institut für Software ) eingesetzt wird.

Eine Server-Applikation bestehend auf vorhandenen und ggf. neuen Software-Komponenten auf offener, professioneller Basis hätte verschiedene Vorteile und käme verschiedenen Projekten zu Nutze, z.B. als verlässliche und performante Basis für Websites (Tiles) und Mobile Dienste (Tiles oder Geometriedaten).

### 3.1.2 AUFGABENSTELLUNG

Es soll eine Server-Applikation mit vorhandenen und ggf. neuen Open Source-Software-Komponenten realisiert werden. Es sind soweit sinnvoll OSM- und OGC-Standards (Interfaces, APIs) einzusetzen, namentlich Tiles, WMS und Web Feature Service (WFS):

- Synchronisation (read) von offiz. OSM-Daten nach lokaler Datenbank in wählbaren Zeitabständen (zwischen Tagen und einigen Minuten)
- Sinnvolle Datenbank-Struktur (PostGIS) passend zu OSM
- WMS inkl. Styles (SLD) optimiert für OSM-Daten
- WFS (read) optimiert für OSM-Daten, z.B. mit GeoJSON (nicht GML 3)
- Optional:
  - WFS write
  - Synchronisation (write) nach offiz. OSM-Service

Folgende Software soll erstellt werden:

- Server-Software, getestet (und demonstrierbar) mit verschiedenen Clients.
- Webseite (Webapplikation) mit kurzer Information zum Projekt und Download-Hinweis (ev. Installations-Hinweise) zur Software.

Lieferdokumente (englisch wo angegeben, sonst deutsch):

- Installationsanleitung (englisch).
- Dokumentierte Demo.
- Gesamter compilier-bereiter Sourcecode (englisch) inkl. Programmdokumentation sowie als Download gekennzeichnetes Zip-File mit ausführbarem Bytecode.
- Technischer Bericht und SW-Engineering-Dokumentation, ev. englisch, inkl. kurze High-level-Programmierdokumentation (englisch).
- Allfällige Dokumente gemäss Vorgaben der Abt. I. (Plakat, ‚Abstract‘).

---

### 3.1.3 HINWEISE

- Die Arbeitsweise ist agil, wo sinnvoll mit Unit-Tests. Es wird Wert auf ausgetestete Software und einfache Installation gelegt.
- Eine Bedienungsanleitung ist nur in begründeten Fällen gefordert.
- Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet, d.h. es wird eine Arbeitsleistung von mind. 360 Stunden pro Person erwartet.

---

### 3.1.4 RANDBEDINGUNGEN, INFRASTRUKTUR, TERMINE UND BEURTEILUNG

- Randbedingungen Hardware/OS:
  - Standard-Hardware
  - OS auf Server: Linux
- Software:
  - ANT, Eclipse IDE, (falls Java SE 1.6 und Tomcat)
  - GeoServer
  - OpenLayers
  - PostgreSQL/PostGIS
- Termine und Beurteilung: gemäss Angaben auf [www.i.hsr.ch](http://www.i.hsr.ch).

Rapperswil den 4. März 2009

## 4 TEIL I TECHNISCHER BERICHT

OpenStreetMap (OSM) ist das 'Wikipedia' der Landkarten. Es ist ein Software-Projekt mit dem Ziel, eine frei nutzbare Weltkarte zu erstellen. OSM wächst seit ca. 2007 exponentiell, ähnlich wie Wikipedia vor 5 Jahren. Im Gegensatz zu Google Maps ist OSM zum Teil detaillierter und aktueller und vor allem: Es gehört allen und kann frei (z.B. in Firmen-Webseiten) auch ausserhalb des Webbrowsers genutzt werden, z.B. in Navigationssystemen (GPS) und Mobiles.

Erfasst werden unter anderem Strassen, Flächen, Gebäude und alle denkbaren Point of Interests. Die Daten können dabei von jedem Community Mitglied erfasst werden. Diese werden per GPS-Trac ins System eingelesen und können danach mit Informationen versehen werden. Die von den Community Mitgliedern erfasste Daten werden via Web API in die Datenbank eingefügt.

Die so erfassten Daten werden mittels verschiedenen Strategien zu Bildern verarbeiten (Kacheln oder Tiles genannt). Die generierten Kacheln können über einen Service mittels Koordinaten und Zoomlevel abgefragt werden.

### 4.1 PROBLEMBSTELLUNG

Da sich die Community über sehr starken Zuwachs erfreut, kommen die OpenStreetMap Server an ihre Grenzen. OpenStreetMap stellt eine Möglichkeit zur Verfügung die Daten auf einem eigenen Server zu verwenden. Die Services können dabei selbstständig aufgesetzt und in Betrieb genommen werden. Regelmässig werden Differentielle-Updates der Daten zur Verfügung gestellt. Somit ist es möglich einen unabhängigen OpenStreetMap Server für private oder firmeninterne Zwecke aufzusetzen.

Die von OSM zur Verfügung gestellten Schnittstellen sind in verschiedene Services aufgeteilt, welche jeweils mit unterschiedlichen Strategien und Programmiersprachen integriert wurden.

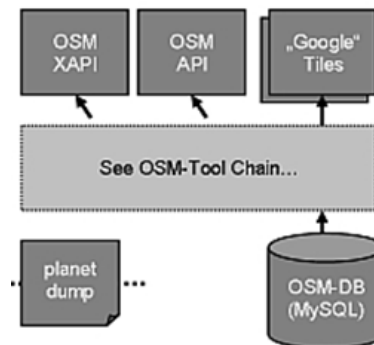


Abbildung 2: OSM-Tool-Chain

## 4.2 ZIEL DER ARBEIT

Das Ziel des Projektes ist die von OSM zur Verfügung gestellten Daten über eine professionelle Schnittstelle der GIS-Welt zur Verfügung zu stellen. Dazu soll eine unabhängige Serverarchitektur erstellt werden, welche die die Daten mit dem öffentlichen OSM System abgleicht und über eine Schnittstelle zur Verfügung stellt. Es ist soweit sinnvoll OSM- und OGC-Standards (Interfaces, APIs) einzusetzen, namentlich Tiles, Web Map Service (WMS) und Web Feature Service (WFS)

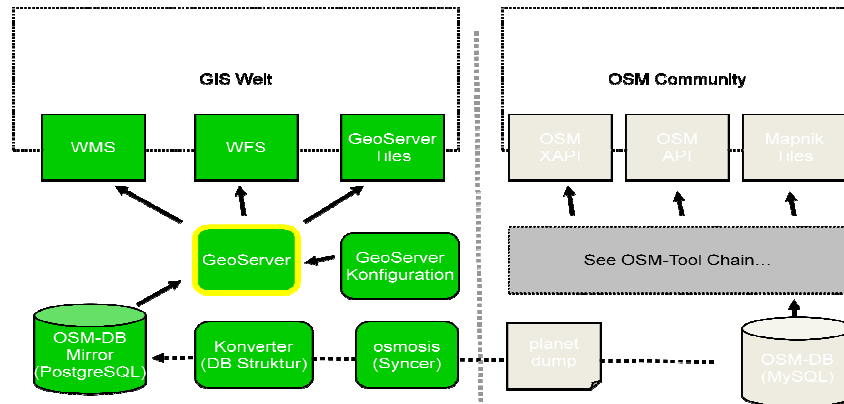


Abbildung 3: Ausgangslage (rechts) mit neuen Erweiterungen (links)

### 4.3 DIE WICHTIGSTEN ANFORDERUNGEN

Funktionalität	Beschreibung/Bemerkungen	Priorität	Abhängigkeit
<b>OSM-Daten Import</b>	Die Daten der öffentlichen OSM-Datenbank sollen auf möglichst effiziente Weise in die lokale Datenbank importiert werden können.	Hoch	
<b>OSM-Daten Aktualisierung</b>	Da sich die Daten in der OSM-Datenbank laufend ändern und neue Daten erfasst resp. modifiziert werden, müssen diese in regelmässigen Abständen überprüft und in der lokalen Datenbank aktualisiert werden. Das Intervall, in welchem die Daten aktualisiert werden soll konfigurierbar sein.	Hoch	Daten Import
<b>OSM-Daten Konvertierung</b>	Damit die von OSM zur Verfügung gestellten Daten vom GeoServer effizient genutzt werden können muss die Datenbankstruktur der lokalen Datenbank entsprechend angepasst werden. Somit ist es notwendig die von OSM erhaltenen Daten in eine andere Struktur umzuwandeln. Dies wird zusammen mit dem Import umgesetzt.	Hoch	Daten Import
<b>GeoServer WFS read</b>	Für die Generierung der Tiles soll eine eigene Strategie entwickelt werden, welche mithilfe des GeoServers realisiert werden soll. Die so generierten Tiles sollen mithilfe des GeoServers im Web bereitgestellt werden. Der GeoServer bezieht die GIS-Daten aus der lokalen Datenbank. Zusätzlich soll der GeoServer die von OSM importierten Daten über die Schnittstellen WMS und WFS zur Verfügung stellen.	Hoch	Daten Konvertierung
<b>Website (WMS Client)</b>	Es soll eine Website erstellt werden welche die Funktionalität des GeoServers demonstriert. Dies soll mithilfe eines WFS-Clients realisiert werden (z.B. OpenLayers) Zusätzlich soll die Website eine kurze Einführung in das Projekt geben, die Möglichkeit bieten die aktuelle stabile Version des Projektes herunterzuladen und eventuell Installationshilfe bieten.	Hoch	GeoServer WMS read

## 4.4 ERGEBNISSE

### 4.4.1 EINBETTUNG DES OSM-IN-A-BOX SERVERS

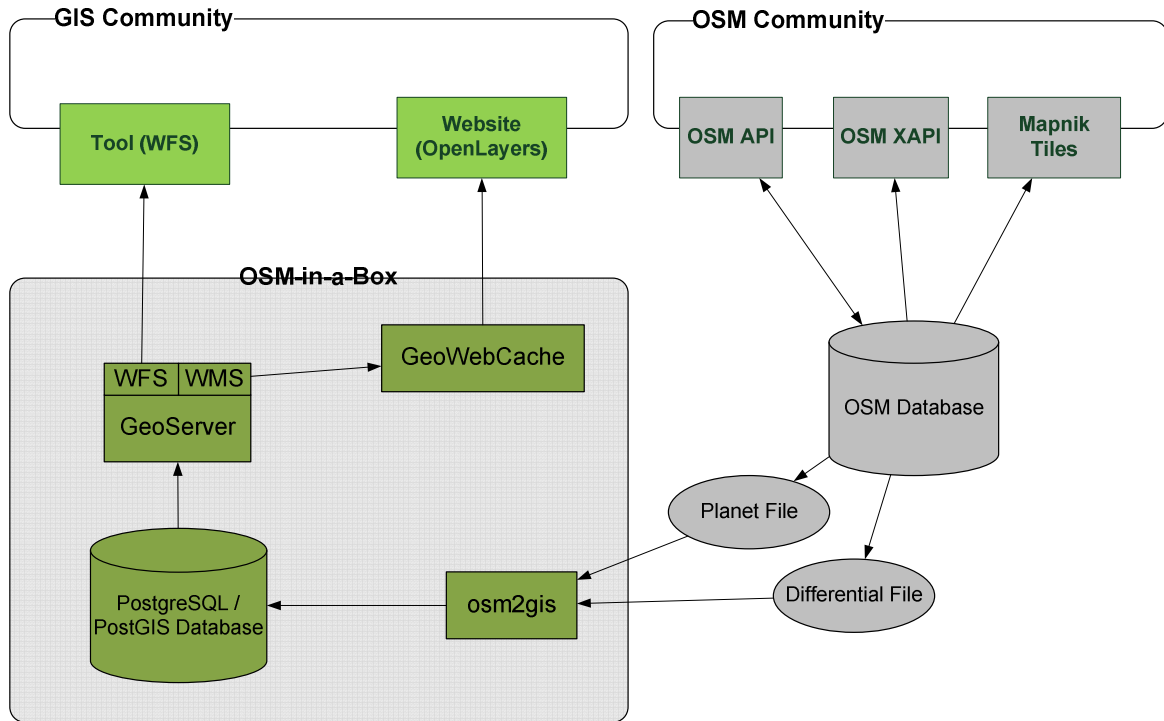


Abbildung 4: Ausgangslage (rechts) mit neuen Erweiterungen (links)

Komponente	Beschreibung/Bemerkungen
<b>OSM Database</b>	Die von in OSM erfassten Daten werden in dieser Datenbank abgelegt.
<b>OSM API</b>	Das OSM-API ist eine Schnittstelle über welche die OSM-Daten abgefragt und verändert werden können. Diverse Kartenbearbeitungswerkzeuge verwenden dieses API um auf die OSM-Daten zuzugreifen.
<b>OSM XAPI</b>	Das XAPI ist eine Ergänzung zum OSM-API, welches verbesserte Abfragen und Suchoption zur Verfügung stellt. Das XAPI ist read-only.
<b>Mapnik Tiles</b>	Mapnik ist ein externes Programm welches aus den OSM-Daten Kartenausschnitte (Kacheln oder Tiles) generiert.
<b>Planet File</b>	Das Planet-File beinhaltet sämtliche OSM-Daten im XML Format beschrieben. Planet-Files werden von OSM wöchentlich generiert.
<b>Differential File</b>	Täglich, stündlich oder minütlich generierte XML Dateien von OSM welche die Änderungen innerhalb dieser Zeitspanne wiedergeben.
<b>PostgreSQL / PostGIS Database</b>	Die PostgreSQL-Datenbank ist der zentrale Teil des OSM-in-a-Box Servers. Die OSM-Daten werden darin in einer GIS konformen Struktur abgelegt.
<b>GeoServer</b>	Der GeoServer ist ein Webserver der unter Tomcat läuft. Er bereitet die Daten aus der PostgreSQL-Datenbank auf und generiert die Kartenausschnitte.
<b>osm2gis</b>	Ist eine Java Applikation welche die Daten von einem Planet-File liest, konvertiert und in die PostgreSQL-Datenbank schreibt. Die Applikation wurde im Laufe der Arbeit entwickelt.
<b>WFS</b>	Web Feature-Service ist ein OGC-Standard für die Abfrage von Vektordaten via http.
<b>WMS</b>	Web Map-Service ist ein OGC-Standard für die Abfrage von Kartenausschnitten über http.

<b>GeoWebCache</b>	Cached die Kartenausschnitte, die vom GeoServer generiert werden und reduziert dadurch die Antwortzeit bei der Abfrage von Tiles (Kartenausschnitte).
<b>Tool (WFS)</b>	Dies kann irgendein Tool sein welches via WFS auf die von GeoServer zur Verfügung gestellten Daten zugreift.
<b>Website (OpenLayers)</b>	OpenLayers ist eine JavaScript Bibliothek welches es ermöglicht, auf einfache Weise eine dynamische Karte in eine Website zu integrieren.

#### 4.4.2 OSM2GIS INITIALER IMPORT

Nachdem der OSM-in-a-Box Server aufgesetzt wurde muss die PostgreSQL Datenbank mit den OSM Daten gefüllt werden. Dies übernimmt die osm2gis-Applikation. Es liest die in XML abgelegten Daten vom Planet File und wandelt sie in eine, für den GeoServer verwendbare, Struktur um. Mittels einer optimierten Strategie werden die losen OSM Daten zusammengefügt und in der Datenbank abgelegt.

**Beispiel** einer Kommandozeile für die Ausführung eines Initialen Imports:

```
osm2gis --initial-import -t -l http://planet.openstreetmap.org/planet-090415.osm.bz
```

Das Planet File wird vom OSM Server heruntergeladen und die Applikation importiert den Inhalt in die lokale PostgreSQL Datenbank. Die Option -t weist die Applikation an die nötigen Tabellen in der Datenbank anzulegen falls diese noch nicht existieren.

#### 4.4.3 OSM2GIS UPDATE SERVICE

Damit die Daten auf dem OSM-in-a-Box Server aktuell bleiben müssen die Änderungen der OSM Datenbank regelmässig mit der lokalen PostgreSQL Datenbank abgeglichen werden. Dazu kann ebenfalls die osm2gis-Applikation verwendet werden. Sie kann als update Service gestartet werden, welcher in regelmässigen Abständen die von OSM zur Verfügung gestellten Differential-Files abfragt und die geänderten Daten in die lokale PostgreSQL Datenbank integriert.

**Beispiel** einer Kommandozeile welche ein regelmässiges Update startet:

```
osm2gis --schedule-update -f minutely -b 200906080828-200906080829.osc.gz
```

Dies startet einen update Task, welche jede Minute ausgeführt wird. Die -b Option bestimmt mit welchem differential File begonnen wird. Danach wird immer automatisch auf das nächste Differential File geprüft. Bei einem update werden ausgehend vom letzten Differential File alle nachfolgenden Differential File, bis hin zum aktuellsten, importiert (sogenannte ‚catch-up‘ Funktion).



#### 4.4.4 OSM2GIS ARCHITEKTUR

Aus der Abbildung wird der logische Aufbau der osm2gis Applikation ersichtlich:

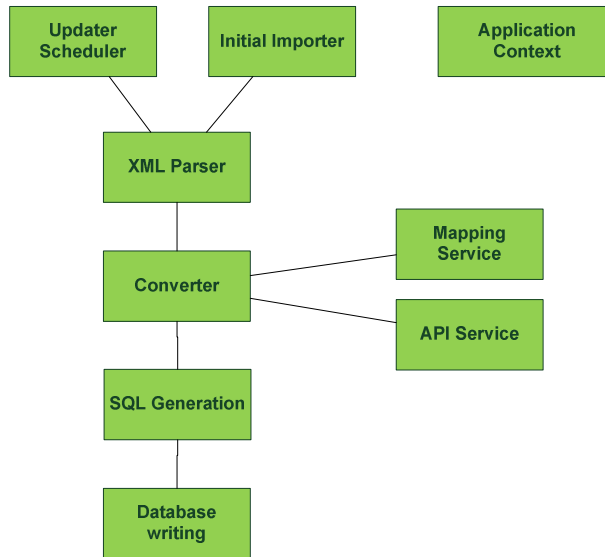


Abbildung 5: osm2gis Software Komponenten

Komponente	Beschreibung/Bemerkungen
<b>Initial Importer</b>	Der Initial Importer führt einen ersten Import der Daten aus. Er kommt zum Zug wenn die Daten zum ersten Mal in die lokale Datenbank importiert werden.
<b>Updater Scheduler</b>	Der Updater Scheduler ist für das regelmässige Ausführen des update Prozess verantwortlich. Er kümmert sich auch darum, dass die Differential Files regelmässig vom OSM Server abgefragt werden.
<b>ApplicationContext</b>	Der ApplicationContext hält die Konfigurationsdaten für den Rest der Applikation bereit. Dies beinhaltet die Daten aus dem Configurations File sowie die der Applikation übergebenen Parameter.
<b>XML Parser</b>	Der XML Parser liest die XML Daten eines OSM Planet oder Differential Files.
<b>Converter</b>	Im Converter Teil der Applikation werden die gelesenen XML Daten für die lokale Datenstruktur aufbereitet.
<b>Mapping Service</b>	Wird vom Converter verwendet um die gelesenen Daten einer Tabelle zuzuweisen.
<b>API Service</b>	Der API Service wird vom Converter dazu verwendet unvollständige Daten über das OSM API abzufragen und zu vervollständigen.
<b>SQL Generation</b>	In diesem Teil der Applikation werden die SQL Statements generiert.
<b>Database Writing</b>	Die Daten werden in die PostgreSQL Datenbank geschrieben.

## 4.5 SCHLUSSFOLGERUNGEN

Das Projekt konnte erfolgreich abgeschlossen werden. Im Verlauf des Projekts wurde ein autonomer Server entwickelt welcher die OSM Daten der Aussenwelt zur Verfügung stellt. Die Applikation osm2gis entwickelt wurde, kann die OSM Daten konvertieren und in die lokale Datenbank importieren. Die Daten werden vom GeoServer dazu verwendet Kacheln zu generieren welche mittels WMS abgefragt werden. Die Vektordaten sind mittels WFS abrufbar.

## 4.6 AUSBLICK

Die osm2gis Applikation funktioniert nur in eine Richtung. Die OSM Daten können importiert, jedoch nicht zurückgeschrieben werden, falls sich die lokalen Daten ändern. Die Applikation könnte so erweitert werden dass die Daten über das OSM API zurück geschrieben werden, was allerdings zu weiteren Problemen führen würde wie: Konsistenzhaltung der Daten. Es müsste eine Strategie entwickelt werden mit der veränderte Daten identifiziert werden können und die OSM IDs (unique identifiers für OSM Daten) korrekt gehandelt werden können.

Grundsätzlich wurde die Applikation so entwickelt dass sie sich möglichst einfach anpassen lässt. Wenn sich in Zukunft an dem OSM Datenschema etwas ändern sollte, oder wenn das Planet File XML Format geändert wird, sollte sich die Änderung an dem osm2gis Applikation in Grenzen halten. Für weitere Details im Bezug auf die Architektur kann das Software-Architektur-Dokument zu Rate gezogen werden.

## 5 TEIL II SW-PROJEKTDOKUMENTATION

### 5.1 ANFORDERUNGSSPEZIFIKATION

#### 5.1.1 ANFORDERUNG AN DIE ARBEIT

##### AUSGANGSLAGE

OpenStreetMap (OSM) ist das 'Wikipedia' der Landkarten. Es ist ein Software-Projekt mit dem Ziel, eine frei nutzbare Weltkarte zu erstellen. OSM wächst seit ca. 2007 exponentiell, ähnlich wie Wikipedia vor 5 Jahren. Im Gegensatz zu Google Maps ist OSM zum Teil detaillierter und aktueller und vor allem: Es gehört allen und kann frei (z.B. in Firmen-Webseiten) auch ausserhalb des Webbrowsers genutzt werden, z.B. in Navigationssystemen (GPS) und Mobiles.

Erfasst werden unter anderem Strassen, Flächen, Gebäude und alle denkbaren Point of Interests. Die Daten können dabei von jedem Community Mitglied erfasst werden. Diese werden per GPS-Trac ins System eingelesen und können danach mit Informationen versehen werden. Die von den Community Mitgliedern erfasste Daten werden via Web API in die Datenbank eingefügt.

Die so erfassten Daten werden mittels verschiedenen Strategien zu Bildern verarbeiten (Kacheln oder Tiles genannt). Die generierten Kacheln können über einen Service mittels Koordinaten und Zoomlevel abgefragt werden.



Abbildung 6: Beispiel einer Kachel  
<http://a.tile.openstreetmap.org/13/4291/2881.png>

Die Kacheln können beliebig in Client Applikationen eingebunden werden. Auf der Homepage von OSM selber werden die Kacheln für die Darstellung einer Slippery Map verwendet. Die OSM Daten sind für viele weitere Einsatzgebiete nützlich.



Abbildung 7:  
OpenStreetMap-Daten in einer Webapplikation (links), in einem 'Consumer Navigationssystem' (GPS)

## PROBLEMSTELLUNG

Da sich die Community über sehr starken Zuwachs erfreut, kommen die OpenStreetMap Server an ihre Grenzen. OpenStreetMap stellt die Möglichkeit zur Verfügung, ihre Daten auf einem eigenen Server zur Verfügung zu stellen. Die Services können dabei selbstständig aufgesetzt und in Betrieb genommen werden. Regelmässige Differential Updates der Daten werden von OSM veröffentlicht. Somit ist es möglich einen unabhängigen OpenStreetMap Server für private oder firmeninterne Zwecke aufzusetzen.

Die von OSM zur Verfügung gestellten Schnittstellen sind in verschiedene Services aufgeteilt, welche jeweils mit unterschiedlichen Strategien und Programmiersprachen integriert wurden.

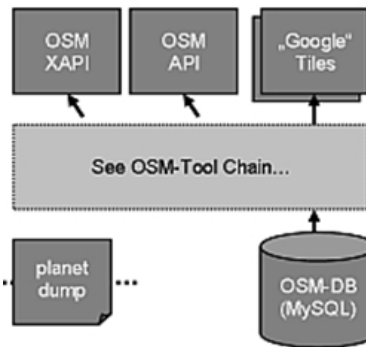


Abbildung 8: Ausgangslage

## ZIEL

Das Ziel des Projektes die von OSM zur Verfügung gestellten Daten über eine professionelle Schnittstelle der GIS Welt zur Verfügung zu stellen. Dazu soll eine unabhängige Serverarchitektur erstellt werden, welche die Daten mit dem öffentlichen OSM System abgleicht und der Aussenwelt zur Verfügung stellt. Es ist soweit sinnvoll OSM- und OGC-Standards (Interfaces, APIs) einzusetzen, namentlich Tiles (Kacheln), WMS und Web Feature Service (WFS)

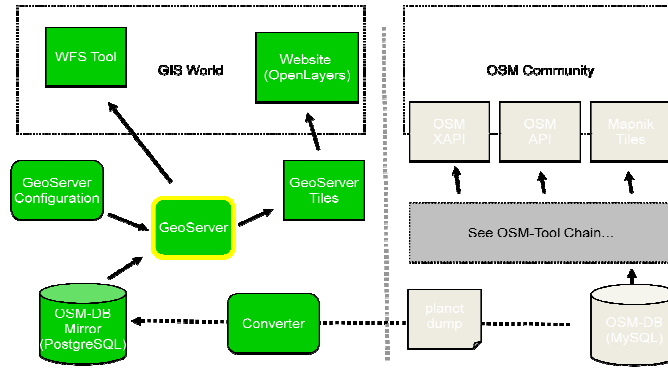


Abbildung 9: Ausgangslage (Rechts) mit neuen Erweiterungen (Links)

Um die Daten aufzubereiten wird der GeoServer verwendet. Wie der GeoServer konfiguriert wird, ist im Verlaufe des Projektes zu evaluieren.

### 5.1.2 FUNKTIONALE ANFORDERUNGEN

Funktionalität	Beschreibung/Bemerkungen	Priorität	Abhängigkeit
<b>OSM-Daten Import</b>	Die Daten der öffentlichen OSM-Datenbank sollen auf möglichst effiziente Weise in die lokale Datenbank importiert werden können.	Hoch	
<b>OSM-Daten Aktualisierung</b>	Da sich die Daten in der OSM-Datenbank laufend ändern und neue Daten erfasst resp. modifiziert werden, müssen diese in regelmässigen Abständen überprüft und in der lokalen Datenbank aktualisiert werden. Das Intervall, in welchem die Daten aktualisiert werden soll konfigurierbar sein.	Hoch	Daten Import
<b>OSM-Daten Konvertierung</b>	Damit die von OSM zur Verfügung gestellten Daten vom GeoServer effizient genutzt werden können muss die Datenbankstruktur der lokalen Datenbank entsprechend angepasst werden. Somit ist es notwendig die von OSM erhaltenen Daten in eine andere Struktur umzuwandeln. Dies wird zusammen mit dem Import umgesetzt.	Hoch	Daten Import
<b>GeoServer WFS read</b>	Für die Generierung der Tiles soll eine eigene Strategie entwickelt werden, welche mithilfe des GeoServers realisiert werden soll. Die so generierten Tiles sollen mithilfe des GeoServers im Web bereitgestellt werden. Der GeoServer bezieht die GIS-Daten aus der lokalen Datenbank. Zusätzlich soll der GeoServer die von OSM importierten Daten über die Schnittstellen WMS und WFS zur Verfügung stellen.	Hoch	Daten Konvertierung
<b>Website (WMS Client)</b>	Es soll eine Website erstellt werden welche die Funktionalität des GeoServers demonstriert. Dies soll mithilfe eines WFS-Clients realisiert werden (z.B. OpenLayers)	Hoch	GeoServer WMS read

	Zusätzlich soll die Website eine kurze Einführung in das Projekt geben, die Möglichkeit bieten die aktuelle stabile Version des Projektes herunterzuladen und eventuell Installationshilfe bieten.		
<b>GeoServer WFS write</b>	Der WFS Standard bietet im Transactional Modus die Möglichkeit neue Daten zu erfassen. Dies soll zusätzlich implementiert werden	Gering	Daten Konvertierung
<b>Daten Synchronisation</b>	Die, über die WFS Schnittstelle erfassten Daten sollen in die öffentliche OSM Datenbank zurück synchronisiert werden.	Gering	GeoServer WFS write

### 5.1.3 NICHT-FUNKTIONALE ANFORDERUNGEN

#### Lokale Datenbank

- Die OSM-Daten sollen in eine Postgres Datenbank abgelegt werden. Um die Struktur der Daten effizient zu gestalten soll das Postgis Plugin von Postgres verwendet werden.
- Die Datenstruktur in der PostgreSQL Datenbank soll nach dem Whitepaper von Jochen Topf[3] übernommen werden.

#### Konnektivität zum OSM Server

- Während der Server läuft muss eine Verbindung ins Internet (bzw zum OSM Server) bestehen, damit der Server die neuen OSM Daten aktualisieren kann.

#### Externe Bibliotheken

- Es sollen möglichst wenig externe Java Bibliotheken verwendet werden um die Abhängigkeiten gering zu halten.

#### GeoServer

- Es sollen die Style Layer Deskriptoren(SLDs) von Mapnik übernommen und für den GeoServer umgeschrieben werden.
- Administrative Boundaries sollen als Linestrings erfasst werden.

#### Auslieferung

- Die Installation soll möglichst einfach zugänglich sein. Dies wird erreicht durch:
- Wenn möglich: Installer verwenden der alle notwendigen Komponente mitliefert.
- One-Click installation.
- Einfach zugängliche Installationshilfe.
- Für alle konfigurierbaren Einstellungen werden sinnvolle Defaults ausgewählt.
- Der Server soll als ein einzelnes Zip File ausgeliefert werden können.

#### Website

- Für die Openlayer Website als Vorlage die Nebelkarte verwenden.
- Sollte mit lat/lon aufrufbar sein. (Permalink).

#### OSM Daten Konvertierung

- Es soll möglich sein den Importer mit einem Cronjob oder dem Windows Scheduler laufen zu lassen.

## 5.1.4 USE CASES

### USE CASE MODEL

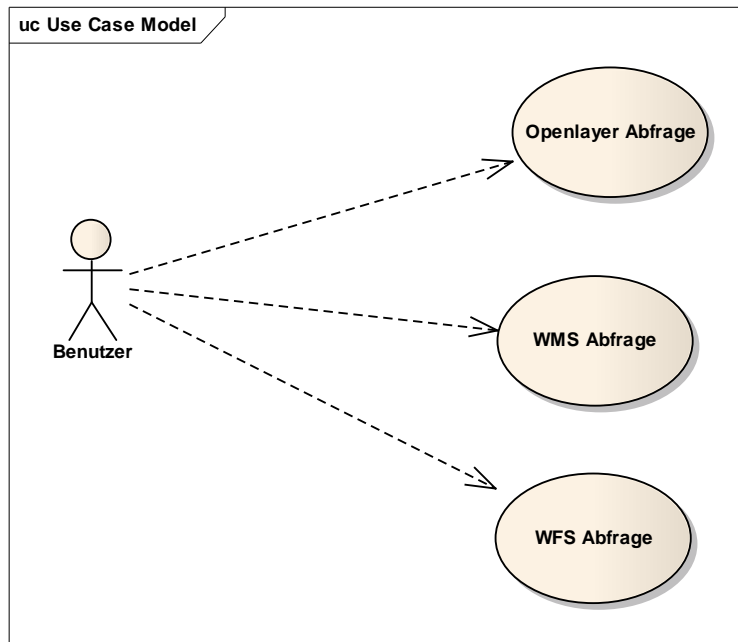


Diagramm 1: Use Case Modell

#### 5.1.4.1 OPENLAYERS ABFRAGE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	<ul style="list-style-type: none"> <li>• Benutzer:             <ul style="list-style-type: none"> <li>○ Der Benutzer möchte den Anfahrtsweg zur HSR einsehen.</li> </ul> </li> </ul>
VORBEDINGUNGEN	-
ERFOLGSGARANTIE	Der Benutzer konnte sich über den Anfahrtsweg an die HSR informieren.
STANDARDABLAUF	<ol style="list-style-type: none"> <li>1. Der Benutzer ruft die Seite mit dem Anfahrtsplan auf.</li> <li>2. Dem Benutzer wird die Anfahrtskarte dargestellt. Er kann diese beliebig vergrößern, verkleinern sowie verschieben.</li> </ol>

ERWEITERUNGEN	Keine
LISTE DER TECHNIK-UND DATENVARIATIONEN	<ul style="list-style-type: none"> <li>• OSM-in-a-Box</li> <li>• Geoserver</li> <li>• Openlayer</li> </ul>
HÄUFIGKEIT DES AUFTRETENS	Wird im Monat mehrmals pro Tag durchgeführt.

#### 5.1.5 WMS ABFRAGE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	<ul style="list-style-type: none"> <li>• Benutzer:             <ul style="list-style-type: none"> <li>○ Der Benutzer möchte via OpenLayers (eingebettet in eine Website) eine Kartenansicht von Rapperswil öffnen.</li> </ul> </li> </ul>
VORBEDINGUNGEN	Der Benutzer hat ein Tool (Website mit OpenLayers) um den Zugriff zu ermöglichen.
ERFOLGSGARANTIE	Der Benutzer konnte die vom Server geladenen Daten graphisch dargestellt ansehen.
STANDARDABLAUF	<ol style="list-style-type: none"> <li>1. Öffnet OpenLayers (Eingebetet in eine Website) Client und wählt Rapperswil als Ort aus.</li> <li>2. Dem Benutzer wird den Kartenausschnitt von Rapperswil angezeigt.</li> </ol>
ERWEITERUNGEN	Keine
LISTE DER TECHNIK-UND DATENVARIATIONEN	<ul style="list-style-type: none"> <li>• OSM-in-a-Box</li> <li>• OpenLayers</li> </ul>
HÄUFIGKEIT DES AUFTRETENS	Sehr unregelmässig, je nach Aktivität 1-2x pro Woche



### 5.1.5.1 WFS ABFRAGE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	<ul style="list-style-type: none"> <li>• Benutzer:             <ul style="list-style-type: none"> <li>○ Der Benutzer möchte die OSM Daten von Rapperswil und Umgebung in einem Vektorformat erhalten.</li> </ul> </li> </ul>
VORBEDINGUNGEN	<ul style="list-style-type: none"> <li>• Lauffähiger OpenStreetMap-in-a-Box Server</li> </ul>
ERFOLGSGARANTIE	Der Benutzer konnte die vom Server geladenen Daten erhalten.
STANDARDABLAUF	<ol style="list-style-type: none"> <li>1. Der Benutzer greift mittels eine Tool dass WFS unterstützt auf den OSM-in-a-Box Server zu</li> <li>2. Der Benutzer bekommt die Vektordaten für den von ihm angegeben Bereich zurück.</li> </ol>
ERWEITERUNGEN	Keine
LISTE DER TECHNIK-UND DATENVARIATIONEN	<ul style="list-style-type: none"> <li>• OSM-in-a-Box</li> <li>• WFS</li> </ul>
HÄUFIGKEIT DES AUFTRETENS	Wenn im Einsatz, sehr häufige Anfragen an den Server.

## 5.1.6 SEQUENZDIAGRAMM

Die Usecases basieren alle auf Request / Response Protokollen. Daher gilt für jeden in den Anforderungsspezifikationen definierten Use Case das gleiche, nachfolgende Sequenzdiagramm.

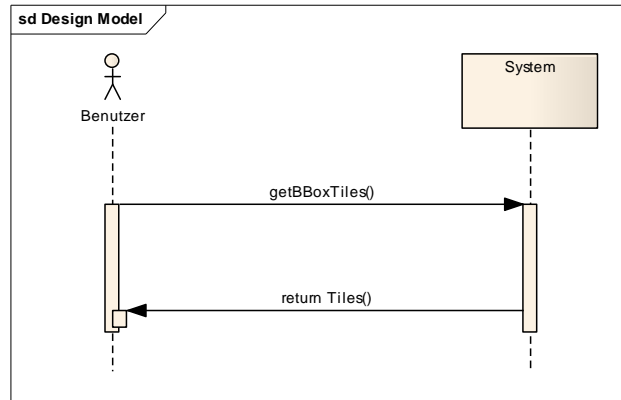


Diagramm 2: Sequencediagramm

### Erläuterungen zum Sequenzdiagramm:

1. Benutzer öffnet eine Website mit OpenLayer.
2. OpenLayers schickt eine Anfrage für ein Tile an das System.
3. Server liefert als Antwort mit einem Tile.  
Dieser Vorgang muss für jedes Tile einzeln erfolgen.

## 5.2 DOMAINMODELL

### 5.2.1 MODELL

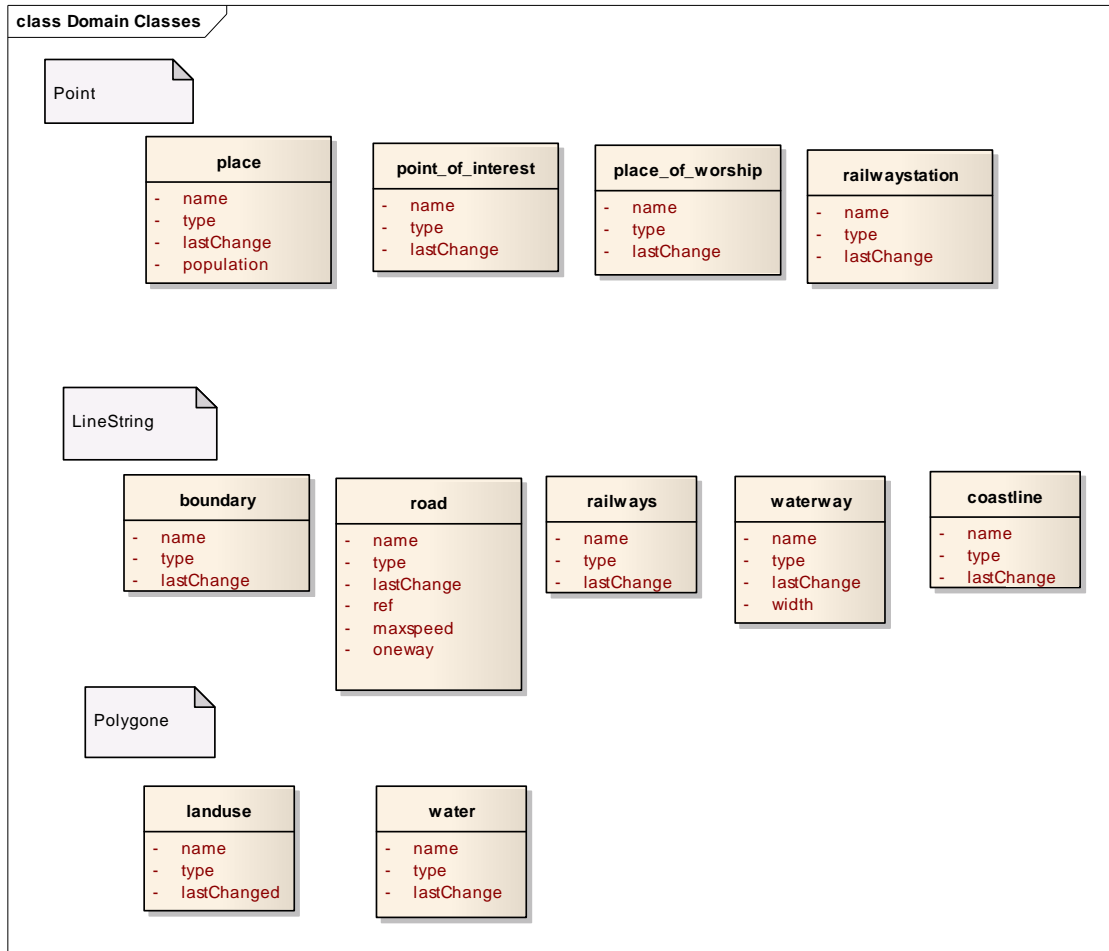


Diagramm 3: Domain Modell

#### Anmerkungen:

Jedes Konzept des unten beschriebenen Domainmodells enthalten die Attribute

- lastChange
- type
- name

## 5.2.2 KONZEPTBESCHREIBUNGEN

Zuerst werden die allgemeinen Attribute, welche alle Konzepte besitzen, erläutert. Anschliessend werden jene Konzepte beschrieben, welche von den Grundlagen abweichen.

### *Alle Konzepte*

<b>Beschreibung</b>	Diese Attribute finden sich in allen Konzepten wieder.	
<b>Attribute</b>	LastChange(Date)	Datum der letzten Änderung
	type(String)	Genauere Beschreibung des Konzeptes. Für Place z.B. ‚Village‘ oder ‚City‘
	Name(String)	Name des Konzeptes.

### *Place*

<b>Beschreibung</b>	Repräsentiert eine Ortschaft.	
<b>Attribute</b>	population(int)	Anzahl Einwohner, falls unbekannt = 0

### *Waterway*

<b>Beschreibung</b>	Repräsentiert eine Wasserstrasse oder Kanal.	
<b>Attribute</b>	width(int)	Weite des Kanals.

### *Road*

<b>Beschreibung</b>	Repräsentiert eine Strasse jeglicher Art.	
<b>Attribute</b>	ref(String)	Referenznummer dieser Strasse. z.B. A1, A5, B542

	maxSpeed(int)	Maximal erlaubte Geschwindigkeit
	oneWay(boolean)	0 = Keine Einbahn, 1 = Einbahnstrasse

## 5.3 DESIGN

### 5.3.1 PHYSICAL ARCHITECTURE

The physical architecture of OSM-in-a-Box can be seen on the diagram on the right.

OSM-in-a-Box downloads the planet or the differential files from an OpenStreetMap-Server and imports them into its own database.

GeoServer, which is a part of OSM-in-a-Box, reads the imported data and create tiles that can be seen in OpenLayers or you can do WMS or WFS requests.

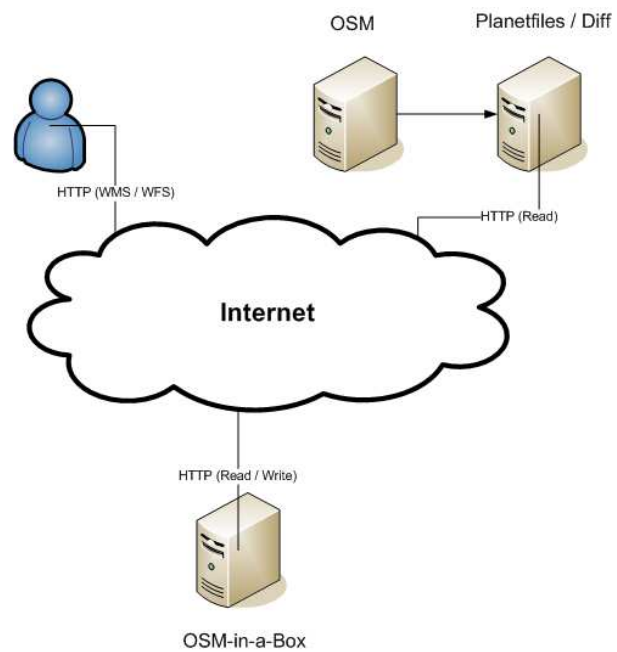


Figure 10: Physical Architecture

### 5.3.2 LOGICAL ARCHITECTURE

Subsequently, the rough overview of the package structure is represented.

The project is divided into the packages you see on the left.



The *parser* package provides the xml parser and its main handler.

The *importer* package contains all handlers and necessary classes for managing the xml parsing process and converting the information into OSM-Entities.

The *db* packages manage the creation of the SQL-Statements for an initial import and the differential updates. The db services are usually called from the package importer. This package is explained in more detail in chapter 5.4.8

Figure 11: Logical Architecture

## PACKAGE DEPENDENCIES

This diagram shows the dependencies between the packages.

The classes stored in the *osminabox* root package are the start classes of this application.

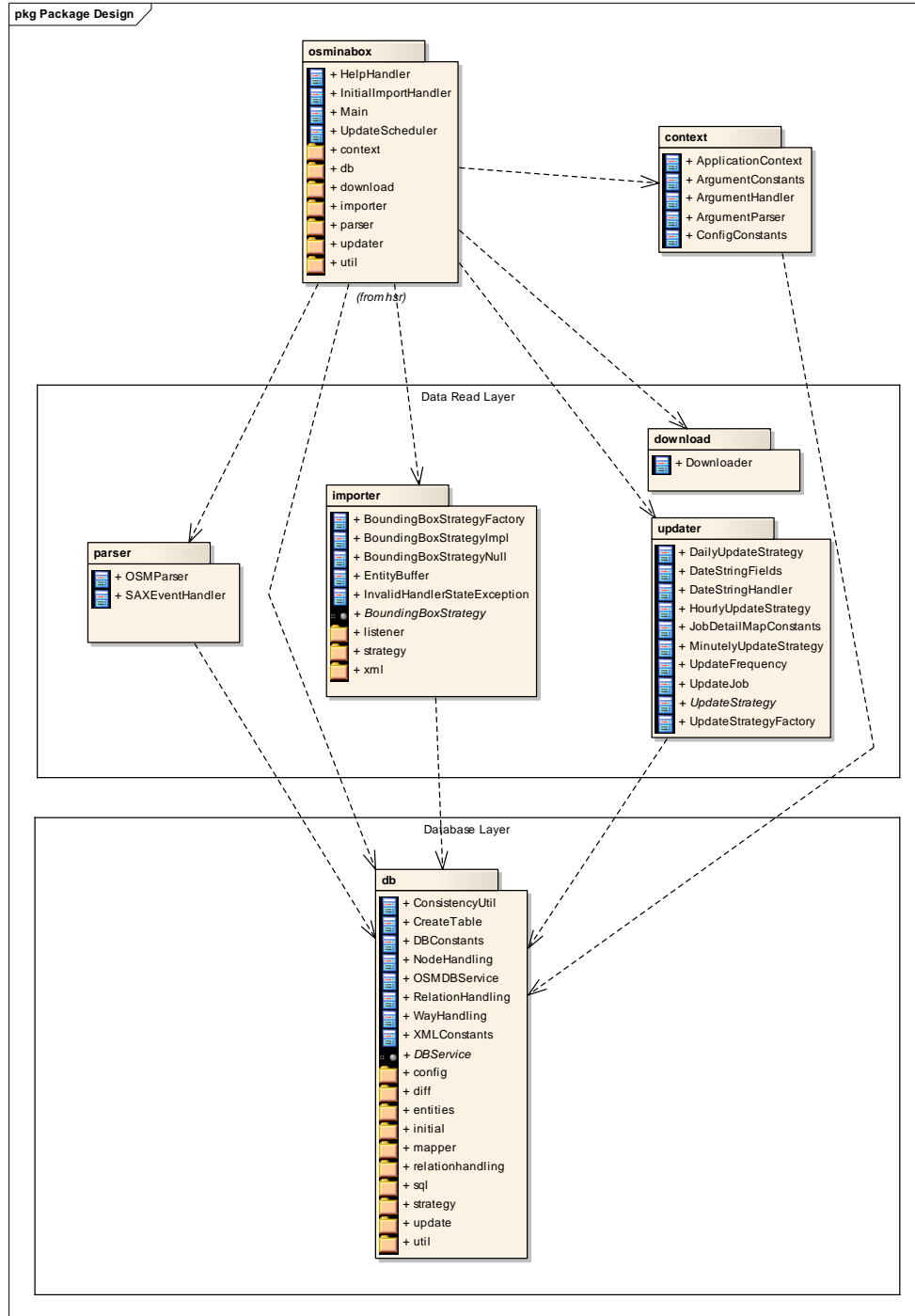


Diagram 4: Package overview

---

### 5.3.3 GENERAL FUNCTIONALITY

The osm2gis application provides two main functionalities:

#### IMPORT OF OSM DATA

The initial import is used to import an OSM planet file into the database. It converts the xml data stored in the planet file into objects and stores them in a table structure made to match the needs of the GeoServer application.

How the initial import is done can be looked up in the user manual.

#### UPDATE OF OSM-DATA

The OSM-Data changes frequently. These changes will be deployed in so called differential files from the OSM-Server. The osm2gis application provides the possibility to schedule an update, which means it will, by itself, fetch the new differential files and import its content into the existing database.

How the updates are scheduled can be looked up in the user manual.

---

### 5.3.4 APPLICATION INITIALIZATION

The main purpose of this chapter is to introduce how the application initialization process works. There are several possibilities to use the application. Therefore, there are parameters triggering the different functionalities of the program. They can be passed to the application via command line arguments.

#### FUNCTIONALITY

The Main class decides which part of the application should be executed and passes the execution on to specific handler classes. These provide the main chain of task execution for a specific functionality.

#### INVOLVED PACKAGES

Packages	Explanation
<b>ch.hsr.osminabox</b>	Entry point of the application / Tasks execution controlling



## IMPLEMENTATION APPLICATION INITIALIZATION

The main classes being part of the application initialization process:

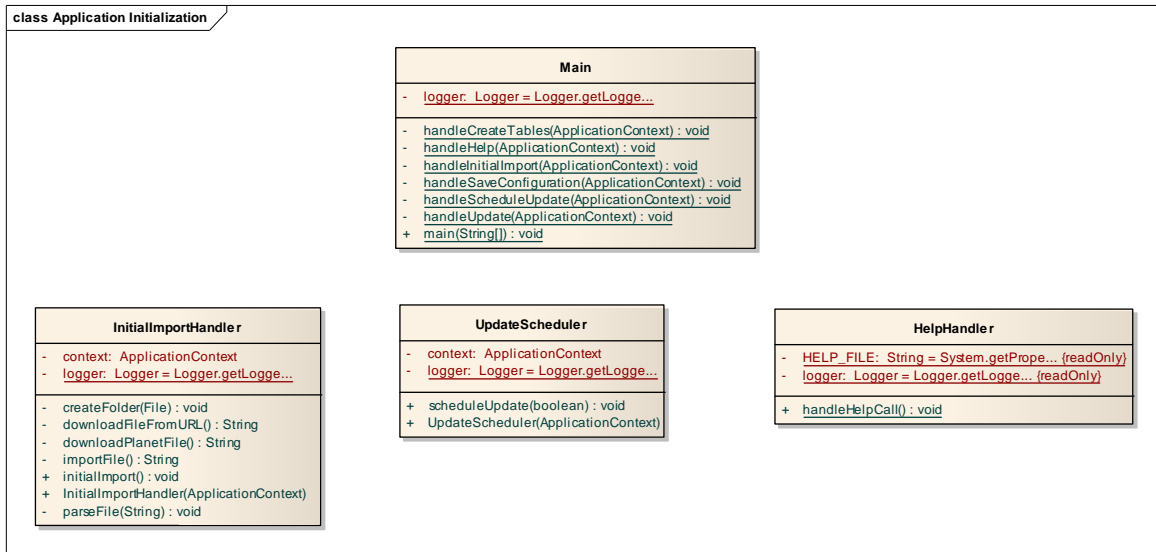


Diagram 5: Main classes

## INITIALIZATION SEQUENCE

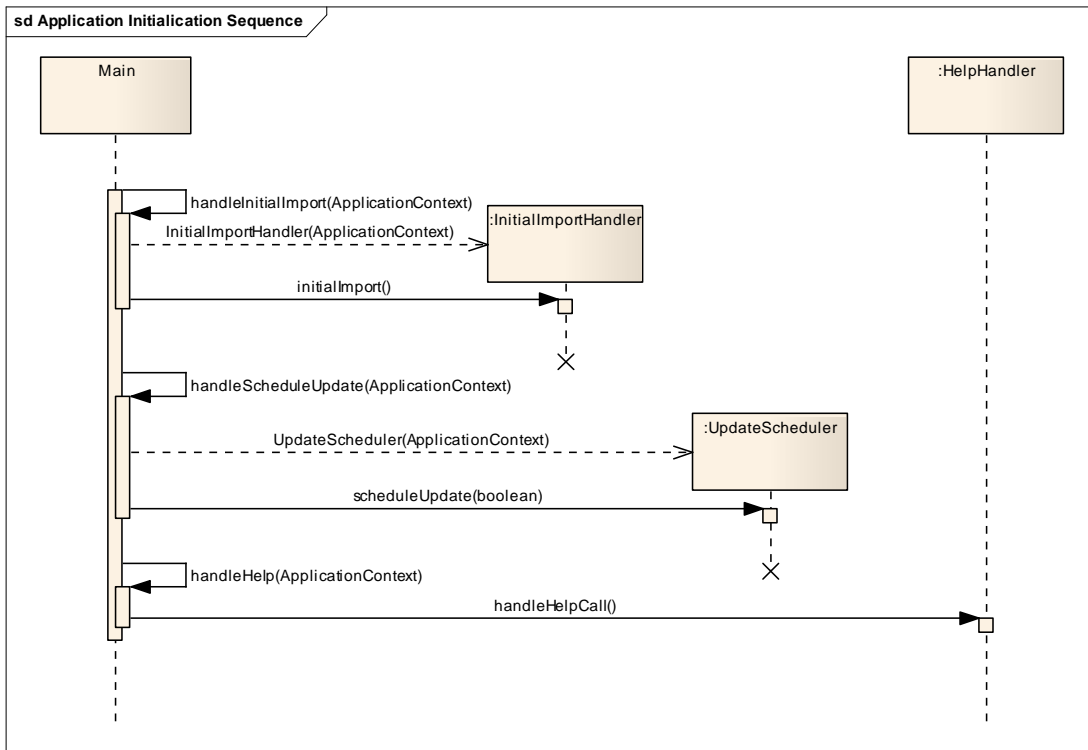


Diagramm 6: Sequece diagram application initialization

## Explanations:

There are three different classes which handle the two core functionalities of the application. The HelpHandler is added to them, to handle a help call.

- InitialImportHandler  
Handle the initial import of OSM planet files.
- UpdateScheduler  
Schedules an update. Updates will be executed frequently. The latest OSM data will be fetched from the OSM file server and the additional data will be merged into the local database.
- HelpHandler  
Prints the 'man' page to the standard output.

The main class decides, based on the arguments passed to the application, which handler will be called.

### 5.3.5 INTRODUCING THE APPLICATION CONTEXT

The ApplicationContext servers the following purpose:

- Provide configuration information from the configuration file
- Provide arguments passed to the application
- Provide a single access point for the database layer

These three information types are needed in different points of the application. In order to avoid global access points and singleton implementation, all the information will be stored in the ApplicationContext. The ApplicationContext will be initialized in the main class and passed to every class which needs access to its functionalities.

#### INVOLVED PACKAGES

Packages	Explanation
<b>ch.hsr.osminabox.context</b>	Contains the ApplicationContext class and some helper classes for argument parsing.

## IMPLEMENTATION APPLICATIONCONTEXT

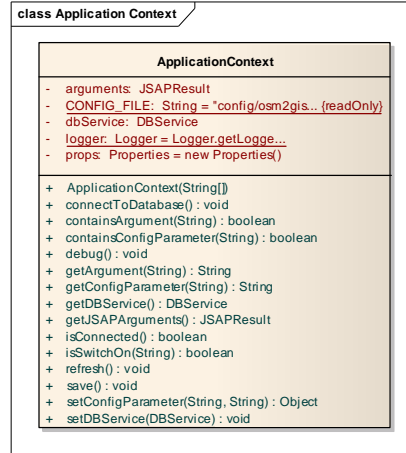


Diagram 7: Application context

### Explanation

The ApplicationContext provides methods to get and set configuration parameters. Calling the save method, all the configuration information will be written to the configuration file.

It also has methods to access the database. The connectToDatabase method will open a database connection using the configuration parameter from the configuration file or the arguments passed to the application if they are set correctly.

### 5.3.6 IMPORTER: PARSING PROCESS

One of the core parts of the application is the parsing process. In most use cases of the application there is an xml file which contains OSM data. This could be a planet file or a differential update file. The xml file is always structured the same way. So it is reasonable to come up with a single parsing process which can handle the differences of the files using different strategies.

## BASIC PARSING ARCHITECTURE DECISION

### Main Issue

To achieve a parsing process which is reliable and scales well, it was necessary to use a streaming parser. The reason for this decision is that the planet files which come from the OSM Database can be very large, more than a hundred gigabytes and growing. It is not possible to read the whole content of the file into the memory and process it afterwards. On the other hand, based on the way the data is stored in the planet file, it is necessary to read data in the first place and combine them with data stored further on in the xml file. There are cross-references in the xml file which need to be resolved before passing them through to the database layer.

### The Osm/OsmChange Format (Issue with streaming Parser)

An exact description of the OSM File format can be looked up on the OSM wiki page [6d]. In this chapter it will only be described in order to explain the architectural decisions that were made, based on that file format.

The OSM data is stored in three different xml elements, nodes, ways and relations. A node is a single point on the OSM map. A way can represent any kind of connection between two or more nodes. The most common use of a way is to describe a smaller or larger road. Relations are there to connect several elements on the map and give them a common meaning. But the only thing the relations are important for, within the application, is to connect ways to an area.

The problem is that the different elements are stored separately in the OSM file format. First all the nodes are listed. After that, the ways are listed, which describes the connections between different nodes. And at last the relations are listed, which can connect several ways. So there is a need to store the nodes in order to process the ways and so on. With the streaming parser technology there is no 'Look back' or no 'Look ahead' at the moment an xml element occurrence is received.

### The solution

Because it is impossible to use a non streaming parser, the 'Look ahead' and 'Look back' functionality has been added within the application. The classes and functionality described in the next section are mainly based on this idea.

## FUNCTIONALITY PARSING PROCESS

The Streaming API for XML (StAX) is an application programming interface to read and write xml documents, originated from the Java programming language. The decision to use this specific xml streaming library was felt because osmosis (An OSM-Tool for converting planet files) uses the same library. Therefore, it is the most reliable library. This chapter describes how the data received from the parser is managed, before it will be passed to the database layer.

## INVOLVED PACKAGES

Packages	Explanation
<b>ch.hsr.osminabox.parser</b>	Contains the main parser classes
<b>ch.hsr.osminabox.importer.</b>	Classes for importer and bounding box strategy
<b>ch.hsr.osminabox.importer.xml</b>	XML Handlers
<b>ch.hsr.osminabox.importer.listener</b>	These listeners are used to act to the events from the XMLHandlers

### 5.3.6.1 TAG HANDLING

#### IMPLEMENTATION ROOT HANDLER

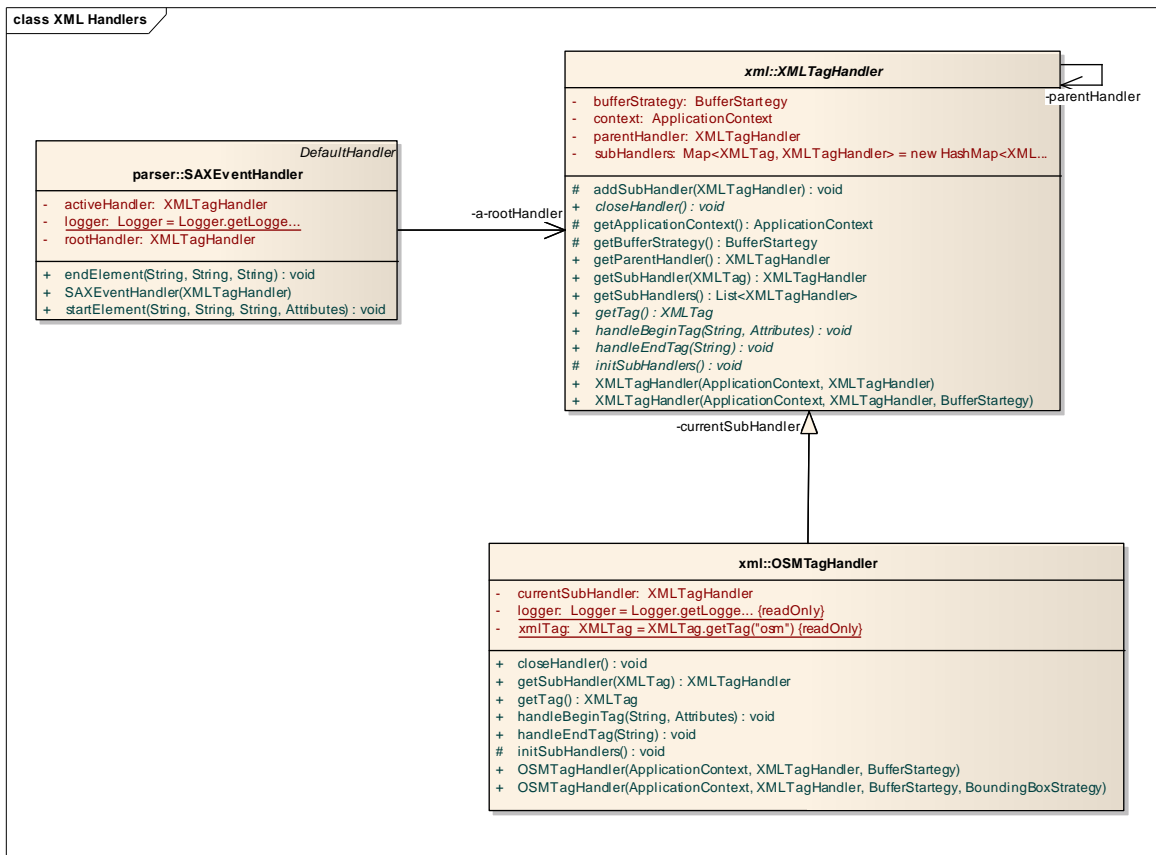


Diagram 8 Class diagram root xml handler

#### Explanation

- SAXEventHandler**  
 The StAX Parser needs a handler to pass the events through. This class is on the top of the parsing process. It receives the occurrence of xml tags while parsing the xml file. If an xml element begins the startEvent method will be called and all xml relevant information will be passed through.
- XMLTagHandler**  
 The XMLTagHandler is an abstract class which provides a basic class to handle an xml element. An implementation of this class handles the occurrence of one specific XML element.
- OSMTagHandler**  
 The OSMTagHandler is the root XMLTagHandler. It is an implementation of the XMLTagHandler and handles the osm element, which is the root element in every xml file generated by OSM.

## TAG HANDLER IMPLEMENTATION

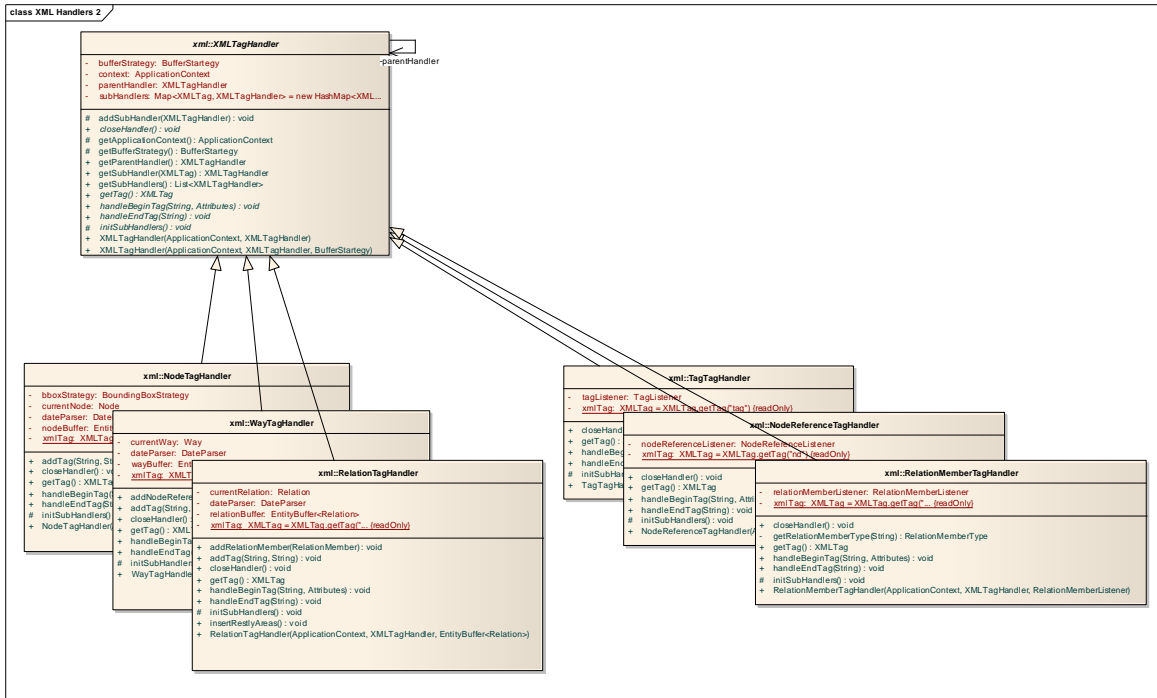


Diagram 9: Tag handler implementation

### Explanation

As said before there is an implementation of an XMLTagHandler for every xml element that occurs in an OSM xml file. The important elements are:

- **Node**  
Holds the values for one specific OSM node
- **Way**  
Holds the values for one specific OSM way, which includes references to nodes.
- **Relation**  
Holds the values for one specific OSM relation, which includes references to ways.
- **Tag**  
Can be a sub element of a node, way or relation element and describes an OSM tag
- **Nd**  
Represents a reference to a node within a way. This can be a sub element of a way.
- **Member**  
Represents a reference to a way or a node within a relation and can be a sub element of a relation.

## HANDLING SUBTAGS

If a streaming parser is used and an xml element occurs, there is no information about the parent element. In this application, the XMLTagHandlers are concatenated hierarchically. Every XMLTagHandler knows his sub- and parent elements. For example, if a tag element occurs, the XMLTagHandler knows his parent XMLTagHandler and passes the tag information to it. The parent XMLTagHandler decides what to do with the information, depending on which xml element is the parent element.

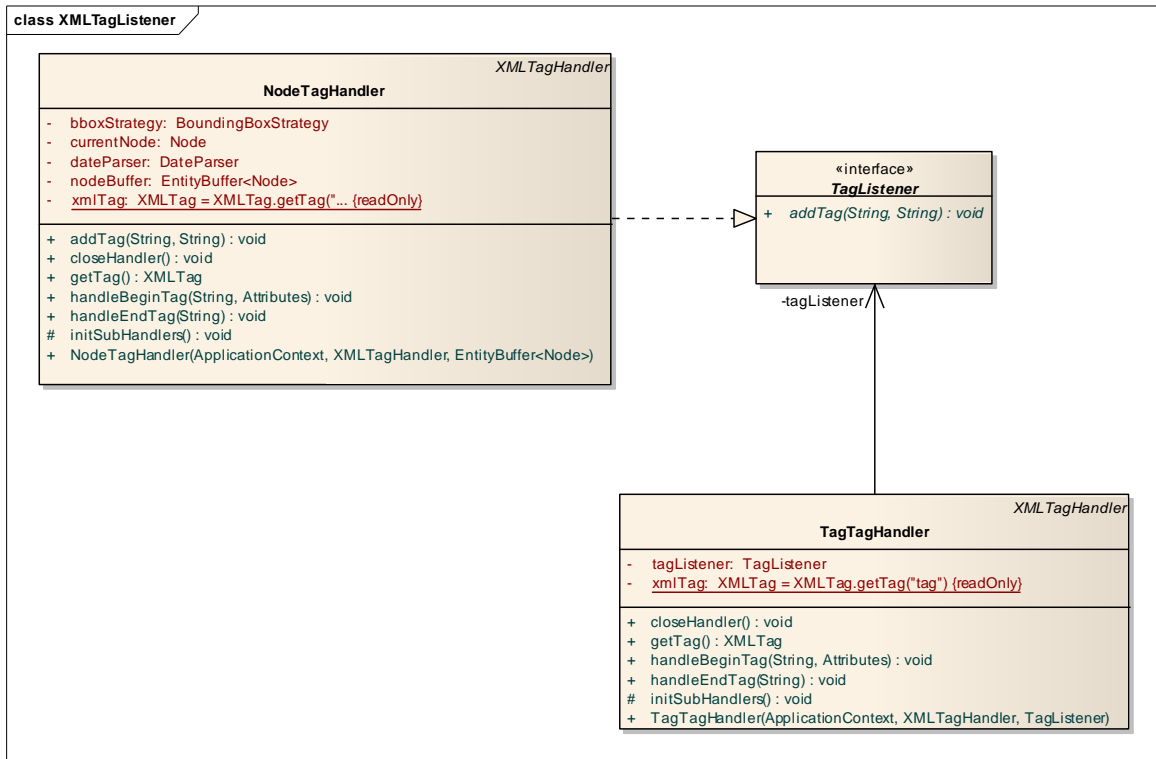


Diagram 10: Class diagram subtag handling

### Explanation

The class diagram shows the involved classes. The NodeTagHandler implements the TagListener interface and passes itself to its sub handler, the TagTagHandler. If the xml element 'tag' occurs, the addTag method on the TagListener will be called and the information will be passed to the NodeTagHandler which adds the tag to the new node object.

## SEQUENCE DIAGRAM TAG HANDLING

The following sequence diagram shows how the TagHandler classes interact with each other to fetch the information from the xml using an example of a node element occurring in the xml file.

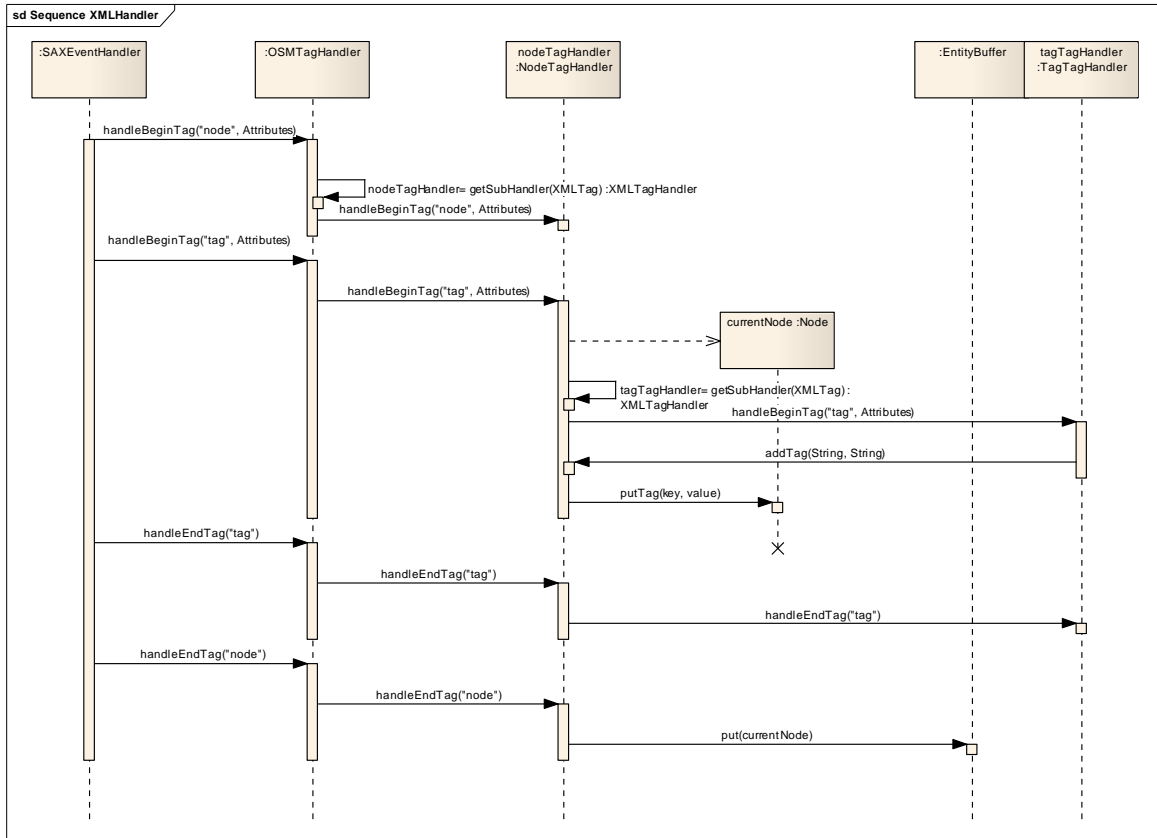


Diagram 11: Sequencediagram tag handling

### Explanation

1. The SaxEventHandler receives a notification that a node element has been read from the xml file.
2. It passes the handling of the occurrence down the XMLTagHandler chain until the responsible tag handler has been found.
3. The NodeTagHandler instantiates a new node element and fills it with the information passed by the xml parser.
4. The node element has a sub element. The SaxEventHandler receives a notification that a tag element has been read from the xml file.
5. It passes the handling of the occurrence down the XMLTagHandler chain until the responsible XMLTagHandler has been found.
6. The TagTagHandler registers the tag and passes its value to the registered NodeTagListener, which in this case is the NodeTagHandler.
7. The NodeTagHandler adds the tag information to the current node object.
8. The SaxEventHandler registers the closing of the tag element. The TagTagHandler receives the close event but does nothing, because no action needs to be taken.
9. The SaxEventHandler registers the closing of the node element. The NodeTagHandler receives the close event and puts the current node on the entity stack.



## CREATING ENTITIES

As described in the chapter before, there is a handler for every xml element that an OSM xml file can contain. For the main elements, such as nodes, ways and relations an entity will be created each time such an element occurs. This entity represents a node, way or relation from the OSM data. This chapter describes what happens to the entities that are created among this process.

## BUFFER STRATEGY

In order to achieve a database usage that scales on a large amount of data, it is not reasonable to insert every entity the moment it occurs. Therefore several buffers are created. The XMLTagHandlers add the new entities to the buffers. If the buffer reaches a given size the content will be passed to the database layer.

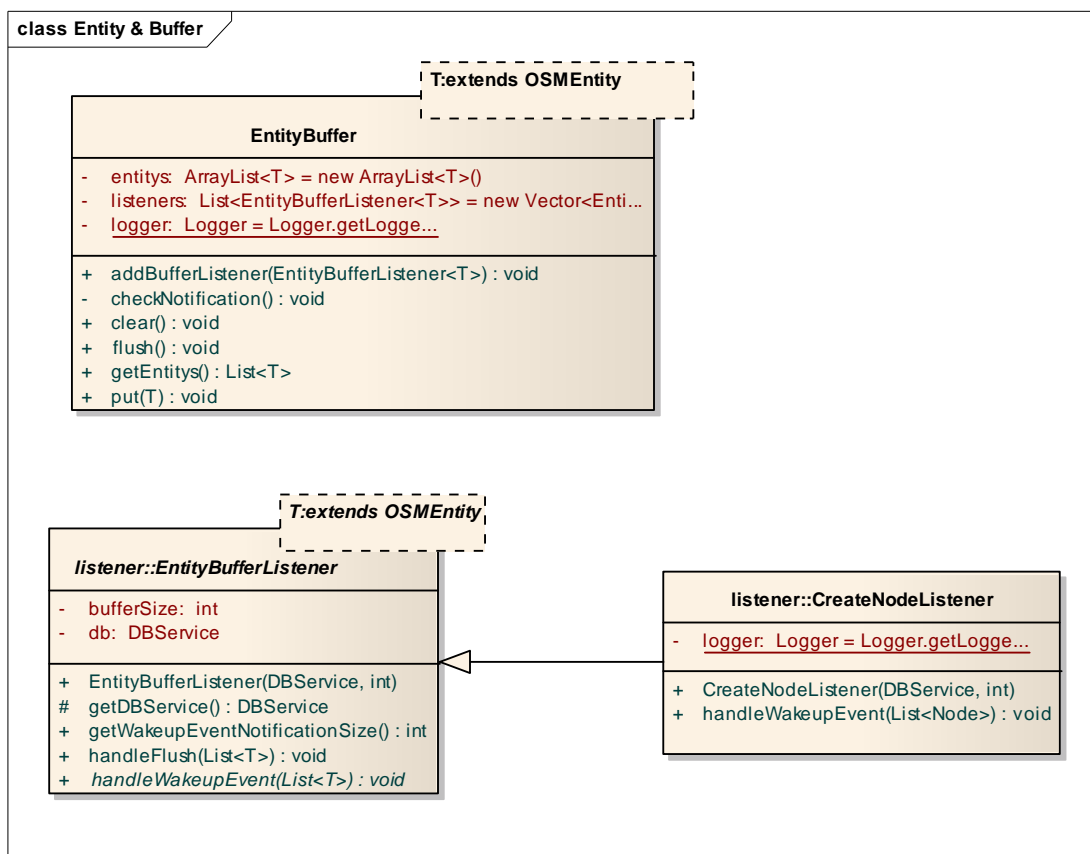


Diagram 12: Class diagram buffer strategy

### Explanation

The diagram shows the EntityBuffer and the EntityBufferListener, which can be registered at the EntityBuffer. The EntityBufferListener specifies the size of the buffer. If the buffer size is reached the EntityBuffer will notify all registered EntityBufferListener calling the handleWakeupEvent and pass its content. The CreateNodeListener is an example of an EntityBufferListener implementation. It will pass the nodes to the database layer by calling the method for inserting nodes on the DBService. The database layer is described in another chapter.

### 5.3.6.2 BOUNDINGBOX STRATEGY

In the most cases there is a planet file which contains the OSM map information for the whole world, but only a smaller area should actually be imported to the database. For those cases, a bounding box can be defined. A bounding box is a rectangle defined by its upper left and lower right corner.

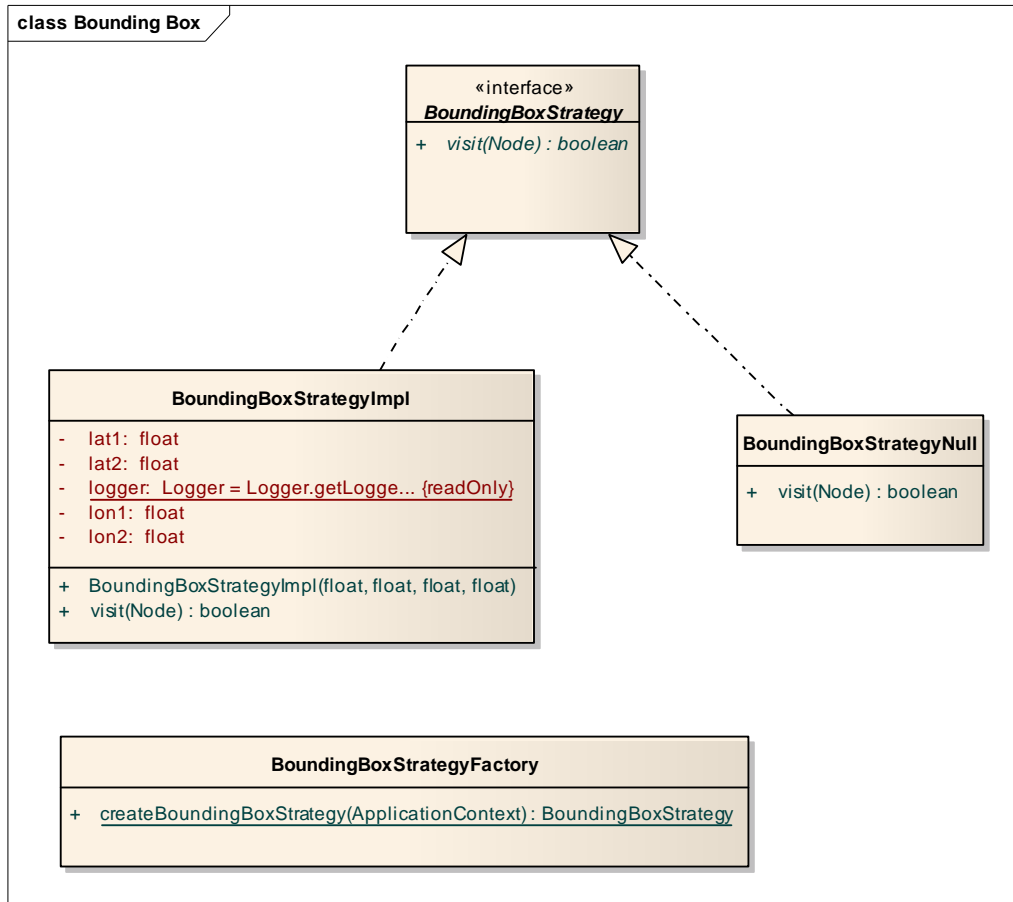


Diagram 13: Class diagram boundingbox strategy

#### Explanation

A **BoundingBoxStrategy** has one method, the visit method. A node can be passed to it and it will return true if the node lies within the specified bounding box. There is also a null object implementation of the **BoundingBoxStrategy** interface. Its visit method will always return true.

### 5.3.7 UPDATE SCHEDULER

In order to keep the OSM data up-to-date, it is necessary to import the differential update files, which are provided by OSM. They can be downloaded from the OSM file server. This chapter describes how the update process is integrated in the application and how the scheduling mechanism works.

#### FUNCTIONALITY

The update process can be initialized passing the right arguments to the application. The files will automatically be downloaded and parsed. Its containing information will be added to the database.

#### INVOLVED PACKAGES

Packages	Explanation
<b>ch.hsr.osminabox.updater</b>	Contains all necessary classes for scheduling the update process

#### IMPLEMENTATION – INTRODUCING QUARTZ

There are several libraries, which provide a suitable task execution framework. In this application Quartz has been used, because it is one of the most known and well documented libraries, therefore reliable. Quartz provides the possibility to easily create tasks and schedules them for a frequent execution.

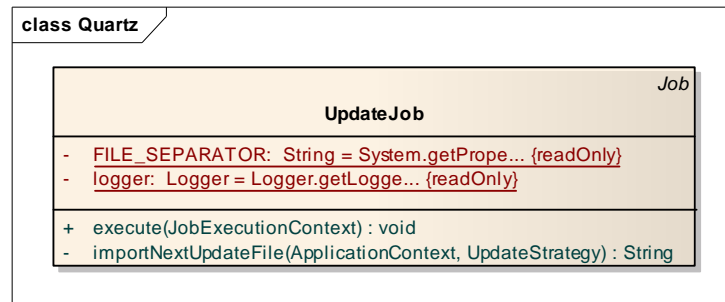


Diagram 14: Class diagram Quartz

To execute a task, there must be a class implementing the Quartz Job interface. The execute method on the job will be called each time the scheduler fires the trigger.

#### IMPLEMENTATION – UPDATESTRATEGY

OSM provides three different update schedules: daily, hourly and minutely. There are two differences which are important for the application in each update scheduling. The names of the differential files changes with every update. To handle these differences without writing same or similar code snippets, the UpdateStrategy will be used.

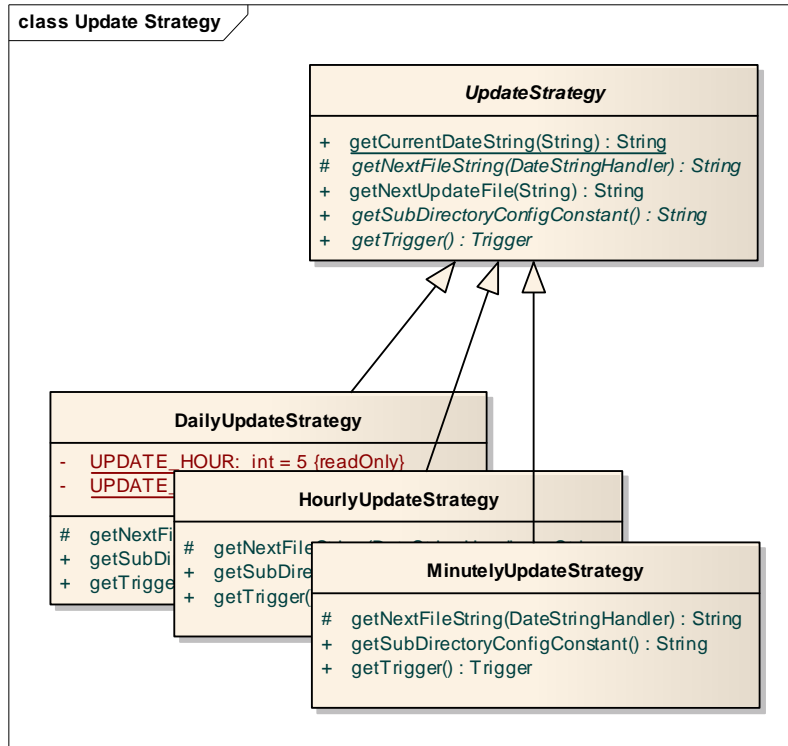


Diagram 15: Class diagram update strategy

Depending on the passed arguments, the corresponding UpdateStrategy will be initialized at the beginning of the application. It provides the scheduler with a trigger which contains the scheduling information. In addition it provides a method for getting the next update file name by passing the current file name. This process is described in the next section.

## CATCH-UP FUNCTIONALITY

This is not an architectural issue, but it is important to mention the catch-up functionality of the update scheduler. It means that if the update process is stopped and restarted again later, the application will remember the last update file and import all files since the stop of the process. This means, that the application can be down for a while, but this will not lead to a loss of update data, if the scheduler I started again later on.

## IMPLEMENTATION – DATESTRING PARSING

The differential update files are stored on a public server. The files will be uploaded frequently. So the updater has to know how the next file is named. The file names have the following file format:

- *Daily*  
[year][month][day-1]-[year][month][day].osc.gz  
Example: 20090525-20090526.osc.gz
- *Hourly*  
[year][month][day][hour-1]-[year][month][day].osc.gz  
Example: 2009052512-2009052513.osc.gz

- *Minutely*  
[year][month][day][hour][minute-1]-[year][month][day][hour][minute].osc.gz  
Example: 200905251230-200905251231.osc.gz

In order to retrieve the next file name there has to be added a minute, an hour or a day to the current file names. This is done by the `DateStringHandler` class.

---

### 5.3.8 DATABASE-LAYER: DATA MIGRATION

This chapter describes the migration of the data given to the database layer from *\*.importer* packages. Due to the size of this chapter, it is divided into the following sub-Chapter.

- General design of database layer
- Initial Import:
  - Node Handling
  - Way Handling
  - Relation Handling
- Differential Update
  - Node Handling
  - Way Handling
  - Relation Handling

## NORMAL PROCEDURE OF DATA IMPORT/ DIFFERENTIAL UPDATE

This diagram shows the normal handling procedure of an initial import or a differential update.

1. First we start the node handling and add every Node.
  - a. If this node is supported, add it to its database table.
  - b. Add to node\_temp database\_table, even if it's not supported!
2. Next, the ways are handled:
  - a. If a way is supported by our application, add it to its database table.
  - b. Add to polygon\_temp table, even if the way is not supported.
3. Afterwards the relations are handled.
  - a. If the relation is supported, it is added to its database table.
4. After the relationhandling is completed, the ways used in the relation are deleted from polygon\_temp database table.
5. As last the polygon\_temp table is scrolled and searched for closed ways, which means, they are areas, and they are added to the database.
6. At last the temporary database tables will be deleted.

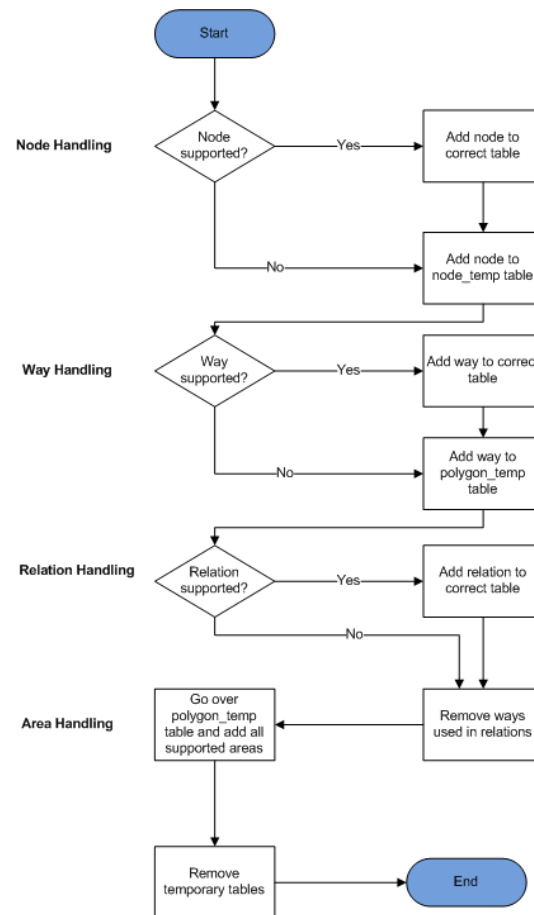


Figure 12: General procedure of data import / differential update

### 5.3.8.1 GENERAL DESIGN

The database layer has one entry point which is an implementation of the DBService interface. The implementation of the interface dispatches all requests that are sent to the database layer.

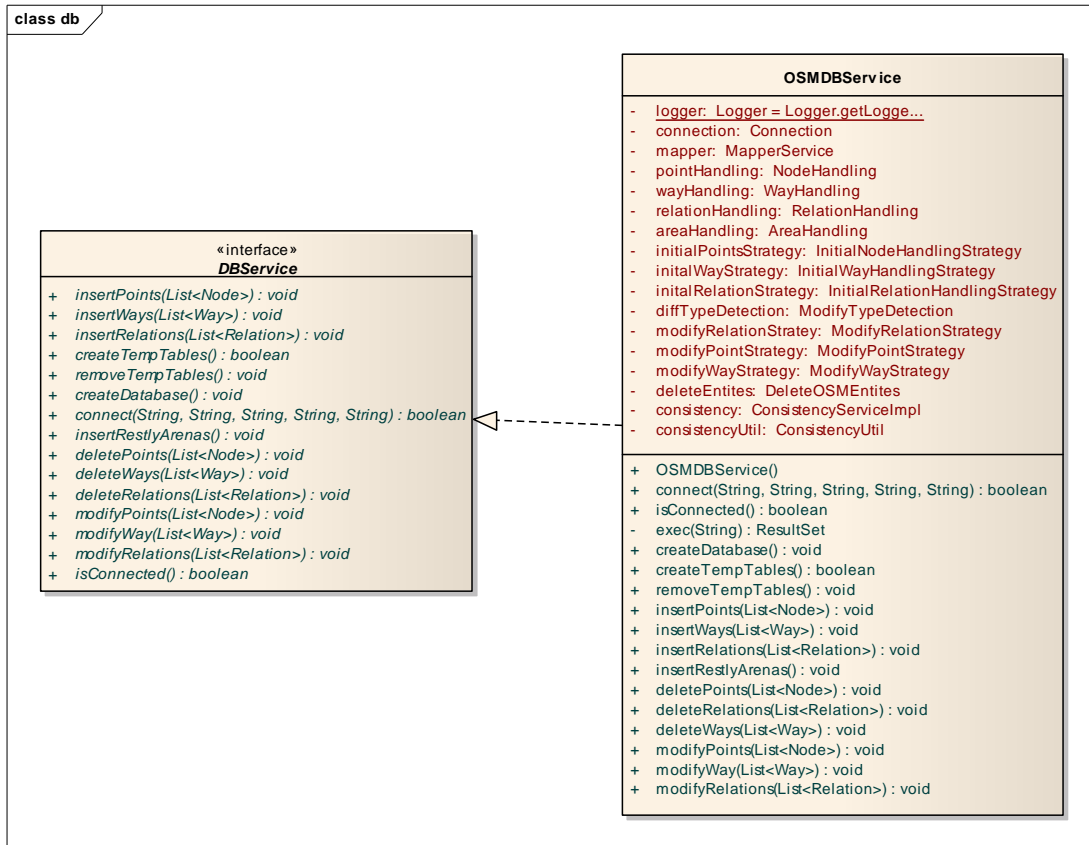


Diagram 16: Class diagram database service

## STRATEGY PATTERN

Due to the complexity of this application a strategy pattern has been added on the top of the application to make it easier to maintain.

### Problem:

- The decision into which database table a node, way or relation has to be added, has to be done for an initial import but also for a differential update. This would lead to duplicated code if no strategy pattern is applied.

### Solution

- The solution was to add a 'global' strategy pattern to the application. The three OSMEntity types (node, way and relation) have their own interface. Two implementations of every OSMEntiys type are created. One for initial import and one for differential update. In the end the 'if, else' code, which decides the database table.

- The strategies are initiated in the OSMDBService and used if needed.

As an example the class diagram for the WayHandling is shown below. These implementations of the WayHandling interface can now be passed to the WayHandling class.

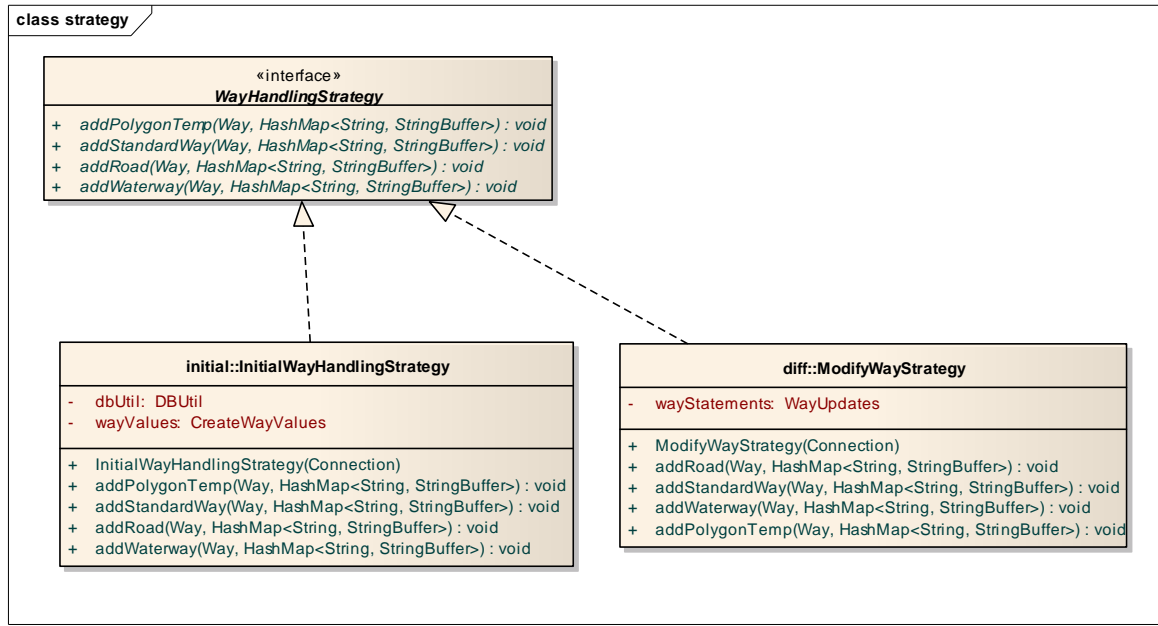


Diagram 17: Way handling strategy example

### 5.3.9 DIFFERENTIAL UPDATE

This chapter explains general aspects of the differential update.

#### MODIFYTYPEDETECTION CLASS

Modifies are differential updates by which the values of an OSMEntity has changed and needs to be updated. Subsequently a list of the possible effects on our database:

- An OSMEntity have **to be inserted**, if the OSM tag was not supported on initial import but has been changed into a Tag that our application supports.
- An OSMEntity **has to be deleted**, if the OSM tag was supported on an initial import, but has been changed to a tag that we do not support.
- An OSMEntity **has to be moved** into a new Database Table, if the OSM tag has been changed to a tag that we still support, but the 'type' has changed .
- An OSMEntity **has to be modified**, if only the values of an OSMEntity have changed.

All these cases are detected in this class and can be handled in the OSMDBService implementation.



## TYPE DETECTION ALGORITHM

This diagram explains the implemented algorithm to detect the type of a differential update. The algorithm can handle all subclasses of OSMEntity.

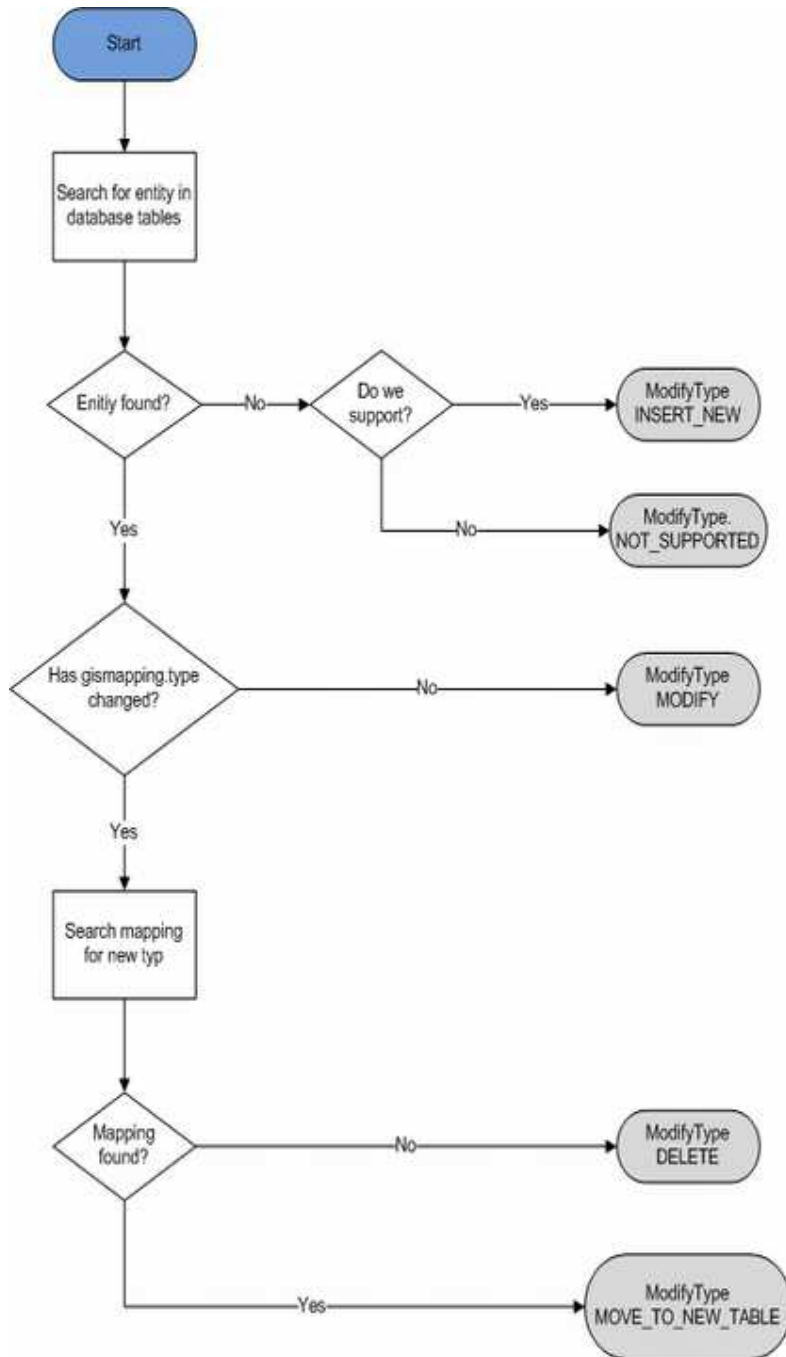


Diagram 18: Differential update type detection

### 5.3.10 NODE HANDLING

#### FUNCTIONALITY

The core functionality is to create SQL insert statements for a given list of nodes. This includes the creation of SQL statements for the different database tables on initial and on differential import. Not all tables have the same attributes.

#### INVOLVED PACKAGES

Packages	Explanation
<b>ch.hsr.osminabox.db</b>	Entry point with method insertPoint(..) and modifyPoint(..)
<b>ch.hsr.osminabox.db.strategy</b>	Node handling strategy
<b>ch.hsr.osminabox.db.initial</b>	Start of the generation of the SQL scripts
<b>ch.hsr.osminabox.db.sql</b>	Creation of the initial node values
<b>ch.hsr.osminabox.db.diff</b>	Creation of the differential values
<b>ch.hsr.osminabox.db.util</b>	Util package, used for HashMap initialization
<b>ch.hsr.osminabox.db.keyvalue</b>	Helperclass for Key/Value array in the database

#### IMPLEMENTATION INITIAL

This diagram shows the implementation for an initial node handling. If the differential handling has to be executed, the InitialNodeHandlingStrategy class has to be replaced with the DiffNodeHandlingStrategy class.

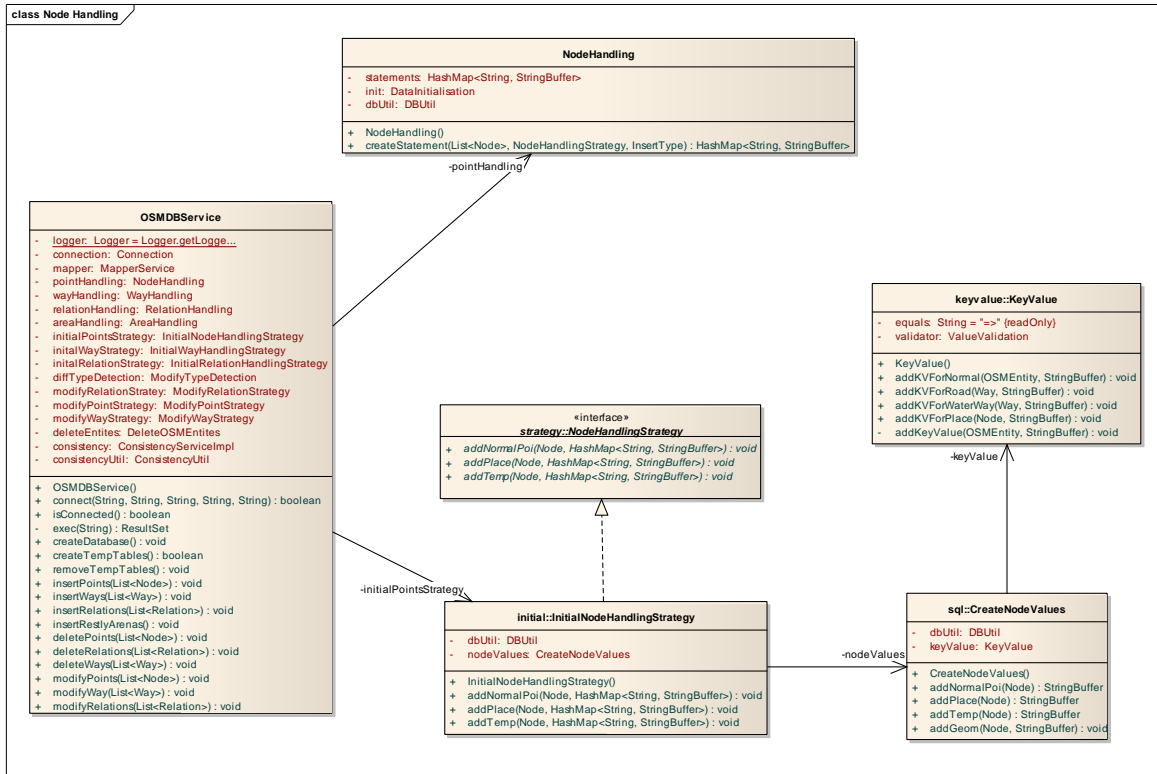


Diagram 19: Class diagram initial node import

## NODEHANDLING CLASS

**Package:** ch.hsr.osminabox.db.NodeHandling

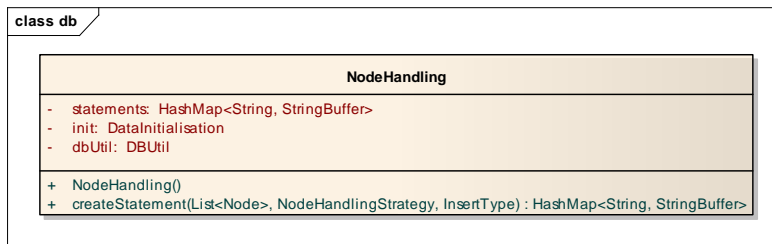


Diagram 20: Class diagram NodeHandling

This is the starting class of the node handling. Both (initial and update) use this class, but they pass their own NodeHandlingStrategy to the createState(..) method.

The returned HashMap contains <String table name, StringBuffer SQL statement>. There is one SQL statement generated for all given nodes.

The first parameter on the createState method is a list containing all nodes. InsertType, the second parameter, is used to define if there has to be added a semicolon at the end of the method or not. For SQL update statements, needed on differential update, it is not possible to update more than one database entity at a time. So a semicolon is added after every update statement. For an initial import, only one semicolon is needed at the end of the SQL statement for all nodes.

## SEQUENCE DIAGRAM INITIAL IMPORT

This sequence diagram shows the handling for a List of nodes on an initial import in a short manner. It is not complete with all details, but it shows the normal process for the node handling.

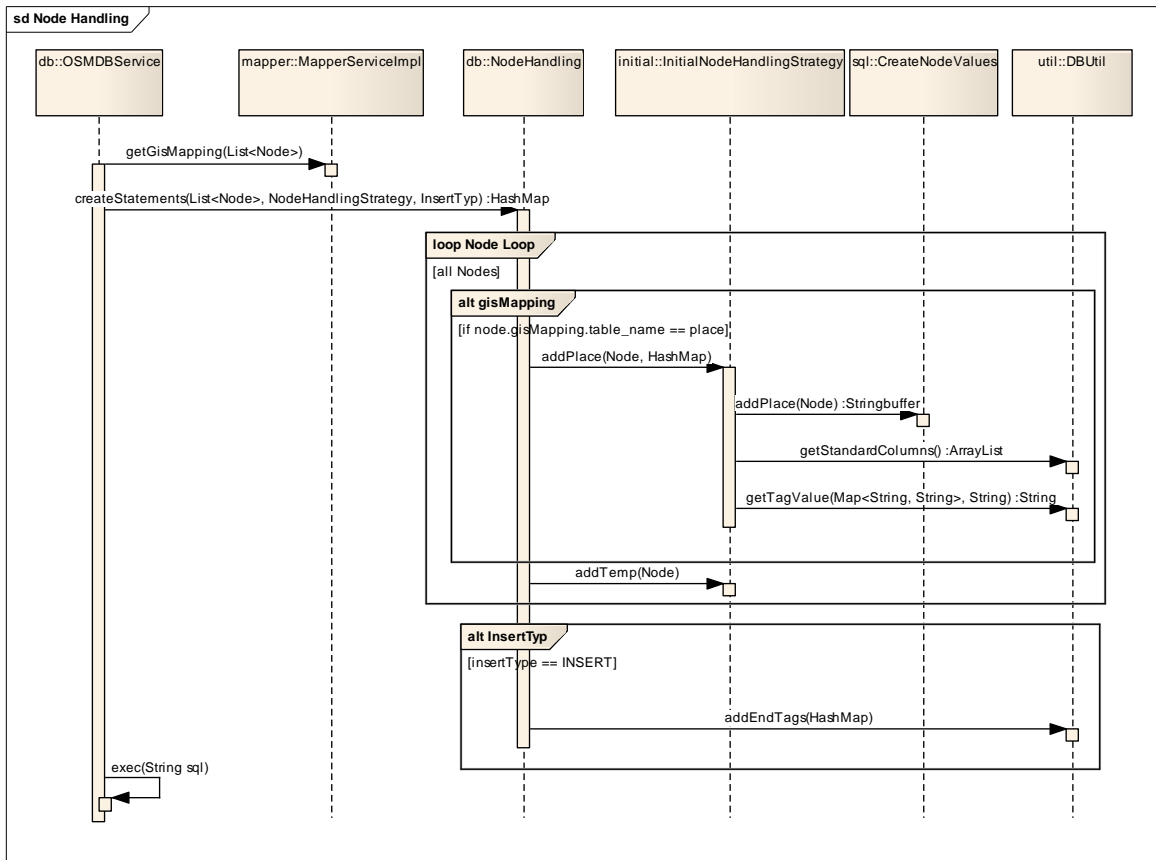


Diagram 21: Sequencediagram initial import

### Explanations

1. The method `insertNodes(List<Node>)` is called.
  - a. OSMDBService calls `getGisMapping`, which adds the `GisMapping` to all nodes in the list. See also the according chapter 5.3.11
  - b. Afterwards, the method `createStatements` on `NodeHandling` is called. As `HandlingStrategy` we pass the `InitialNodeHandlingStrategy`, because an initial import has to be handled. `InsertType` is `INSERT`
2. `NodeHandling` class.
  - a. The `NodeHandling` class loops over all nodes in the node list and also in the list of `GisMappings`. (An `OSMEntity` can be added to more than one database table.)
  - b. The 'if-else' code decides in which table this node has to be inserted. In this case, it is a place. So the `NodeHandling` calls `addPlace(Node, HashMap)` on the `NodeHandlingStrategy`.
3. `InitialNodeHandlingStrategy` class.
  - a. Initializes the SQL statement, if it is not already done by a previous call, and adds the new values to the SQL statements.
  - b. For the table `place`, further attributes are needed in addition to the standard values. They are added by calling `getTagValue` on `DBUtil`.

4. Every node, even if he has no valid gisMapping, has to be added to the node\_temp database table.
5. NodeHandling class.
  - a. After adding all nodes to their according StringBuffers in the HashMap, a semicolon will be added at the end of every statement.
6. OSMDbService now loops over the values of the resulting HashMap and executes every SQL Statement.

## IMPLEMENTATION DIFFERENTIAL UPDATE

The differential handling uses the same strategy pattern as all other handlings. See also Chapter 5.3.8.1

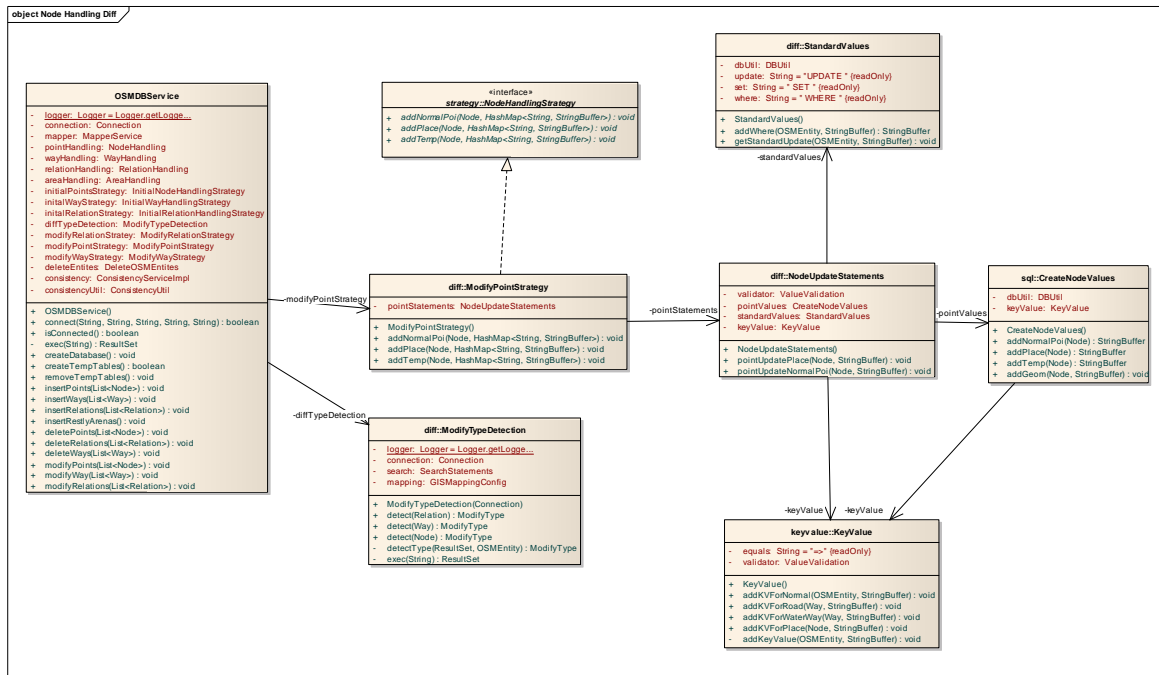


Diagram 22: Class diagram differential update

## Notes

- On the differential update the same `addGeom` function on `CreateNodeValues` will be used as on an initial import. Otherwise this would lead into duplicated code.
- For explanation about the class `NodeHandling` see the chapters above.

### 5.3.10.1 WAY HANDLING

#### FUNCTIONALITY

The core functionality is to create SQL Insert statements of the given list of ways. This includes the creation of SQL statement for the different database tables on initial and on differential import. Not all tables have the same attributes.

#### INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry Point with method insertWay(..) or modifyWay(..)
ch.hsr.osminabox.db.strategy	WayHandling strategy
ch.hsr.osminabox.db.initial	Start of the generation of the SQL scripts
ch.hsr.osminabox.db.sql	Creation of the initial way values
ch.hsr.osminabox.db.diff	Creation of the differential values
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization
ch.hsr.osminabox.keyvalue	Helperclass for Key/Value array in database

#### IMPLEMENTATION INITIAL IMPORT

Classes that are used for an initial import on ways are shown below.

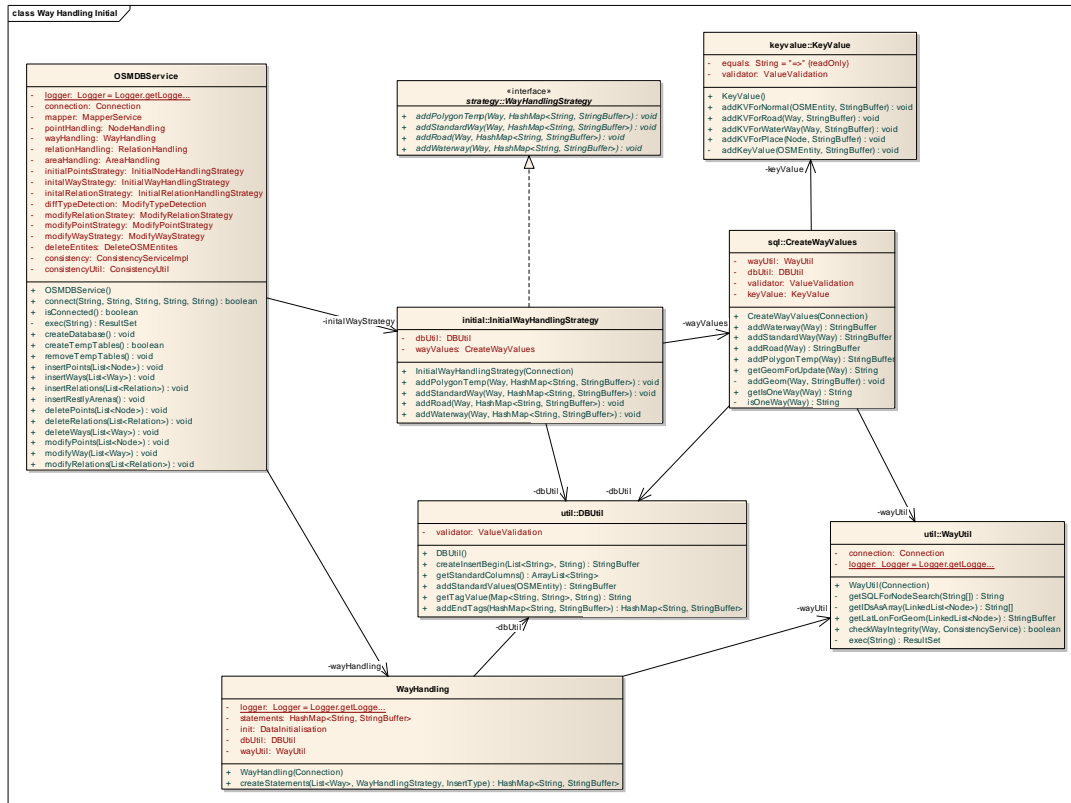


Diagram 23: Class diagram initial way import

## SEQUENCE DIAGRAM INITIAL IMPORT

This Diagram shows the rough process of the initial import on ways

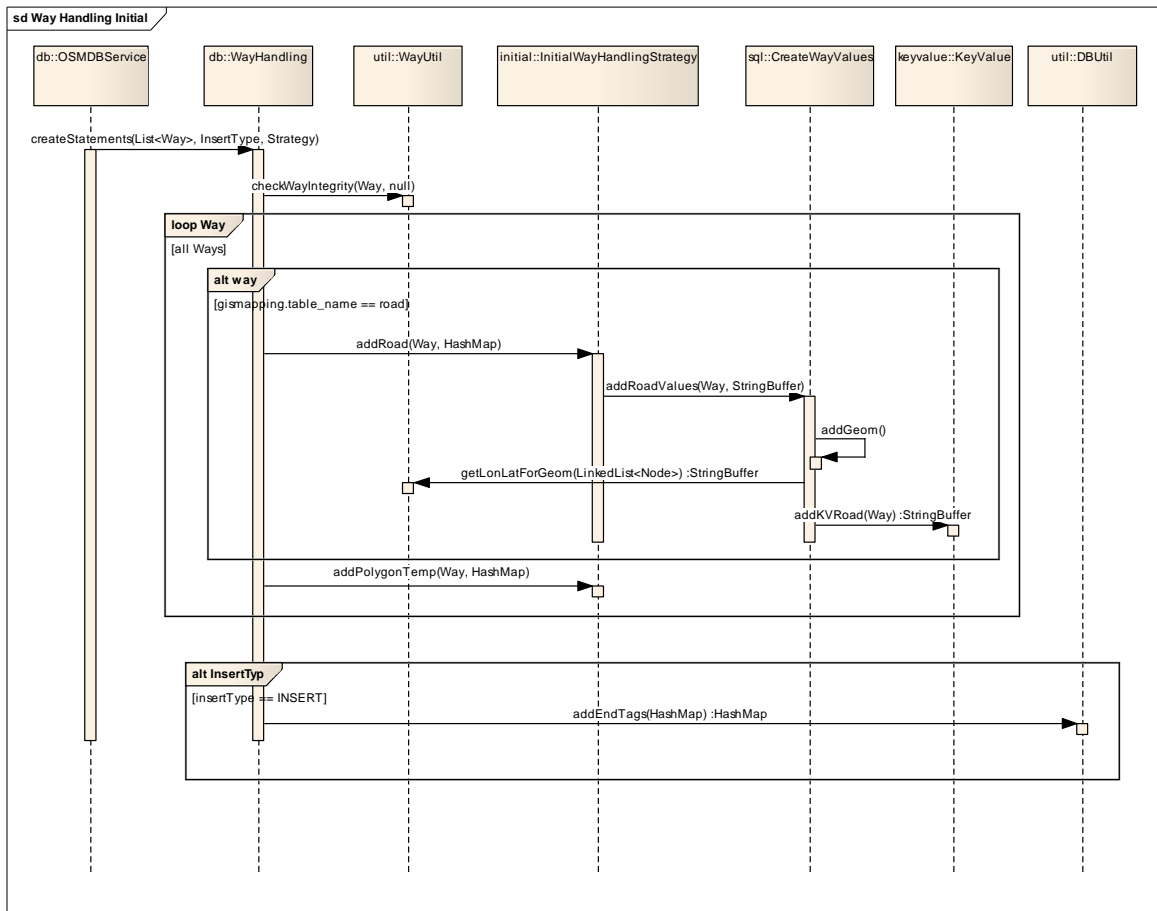


Diagram 24: Sequence diagram initial way import

### Explanations

- 1) \*.importer calls the method insertWays(List<Way>) on OSMDBService.
- 2) OSMDBService calls createStateStatement on WayHandling and passes the according WayHandlingStrategy
  - a) WayHandling checks the way integrity by calling the method checkWayIntegrity on WayUtil. This method needs two parameters. The second parameter is used for consistency for modifyWay(...) which will be described in detail later. For an initial import this parameter needs to be null.
    - i) The method checkWayIntegrity checks on the database if all nodes inside the way are present in the node\_temp table and adds the coordinates of the table to the nodes inside the way.
- 3) After checking the way integrity, the class decides in which table the way has to be added and calls the correlated method on the InitialWayHandlingStrategy.
- 4) The strategy initializes the SQL statement if needed with 'INSERT INTO' and calls addRoadValues() on CreateWayValues.
- 5) CreateWayValues adds the needed values including the geom.
  - a) The addGeom method calls a method on WayUtil to get the lon/lat string for the geom.
- 6) At the end the way is added to the polygon\_temp table, if he is supported by our application or not.

## IMPLEMENTATION DIFFERENTIAL UPDATE

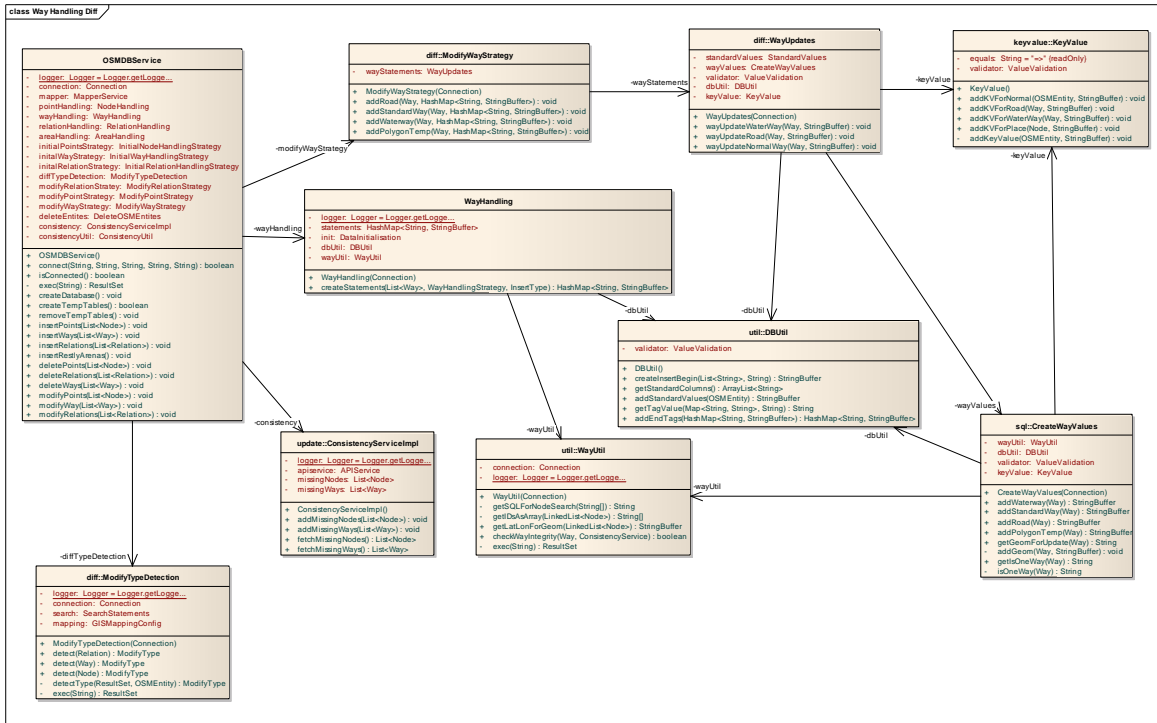


Diagram 25: Class diagram differential way update

### Explanations

- The way differential handling uses ModifyTypeDetection class as well. See also 5.3.9
- Differential update on ways has a specialty due to the format of the differential file:
  - ConsistencyServiceImpl:
    - This service is used for a specialty. The OSM differential files sometimes do not deliver all needed nodes for a way to be modified. But an update in our database needs all information, which means also the containing nodes in a way.
    - The ConsistencyService is needed if a node is missing in the way. The node is then received for OSM online API.

### SHORT WALKTHROUGH ON DIFFERENTIAL WAY UPDATE

Below a short walkthrough of the differential way update.

1. Call of modifyWay(List<Way>) on OSMDbService
  - a. OSMDbService calls wayUtil.checkWayIntegrity() now with the second parameter ConsistencyServiceImpl.
  - b. Typedetection is done.
2. From this point it works similar to initial way handling just with another WayHandlingStrategy.



### 5.3.10.2 RELATION HANDLING

Relation handling is the biggest part of the osm2gis importer inside the database layer.

#### FUNCTIONALITY

Here is a list of the functionality implemented on relation handling.

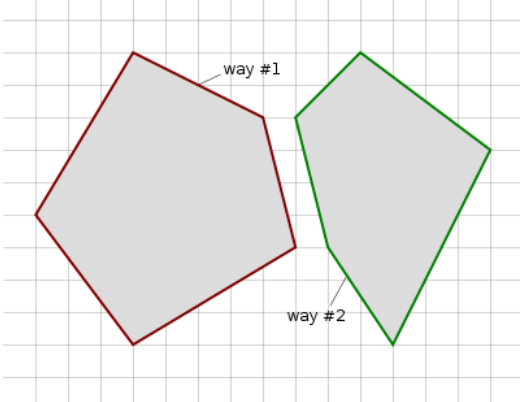
- Creation of SQL statements for initial imports and differential updates.
- Detection of the relation type.
- Special handling of relation areas with inner and outer ways.
- Detection and removal of combined ways.

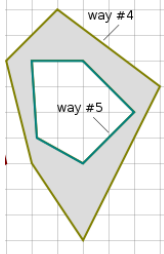
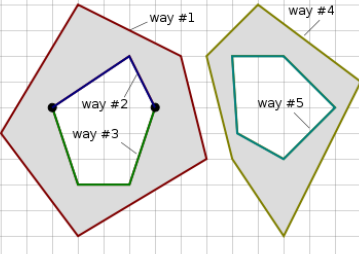
#### INVOLVED PACKAGES

Packages	Explanation
<code>ch.hsr.osminabox.db</code>	Entry point with method <code>insertRelation(..)</code> and <code>modifyRelation(..)</code>
<code>ch.hsr.osminabox.db.strategy</code>	Relation handling strategy
<code>ch.hsr.osminabox.db.relationhandling</code>	Start the creation of SQL scripts and holder of util classes for relation handling
<code>ch.hsr.osminabox.db.sql</code>	Creation of the initial relation values
<code>ch.hsr.osminabox.db.sql.diff</code>	Creation of the differential relation values
<code>ch.hsr.osminabox.db.diff</code>	Creation of the differential values
<code>ch.hsr.osminabox.db.util</code>	Util package, used for HashMap initialization
<code>ch.hsr.osminabox.db.keyvalue</code>	Helper class for Key/Value array in database

#### RELATION TYPES

Given of the relation data type from OpenStreetMap we have to support three types of relations

1.	<p>The first type is <b>only outer</b>. This means that the relation has only ways that are outer boundaries of a polygon.</p>	 <p>Figure 13: only outer relation</p>
----	--	--

<p>2.</p>	<p>The second type has <b>one outer</b> and contains <b>several inner</b> ways. Normally this can be used to model a big area with holes.</p>	 <p>Figure 14: one outer, several inner relation</p>
<p>3.</p>	<p>The third type is the combination of type 1 and 2. This type can <b>have several inner and outer ways</b>. The specialty of the left area with 2 inner ways will be explained in the next chapter.</p>	 <p>Figure 15: n outer, n inner relation</p>

## COMBINED WAYS

OpenStreetMap implemented some specialties. One of them is that a boundary of a polygon can be formed by more than one way. What was thought as a simplification, turned out to be much more complicated. The interpreting application has to search itself for these combined ways. There is nothing tagged inside the relation that would make it clear that these ways are combined.

For this feature we had to implement an algorithm that removes this combined ways and consolidates them into one big way because PostGIS is not able to handle boundaries of areas which are not one line.

Here we have some examples of combined relations. One you have already seen in the chapter above.

An easy example would be this:

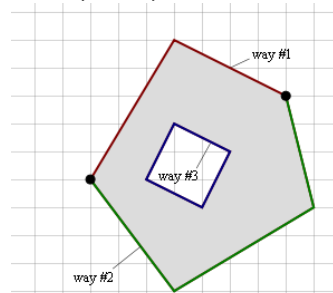


Figure 16: Combined way example 1

This of course can be carried to extreme.

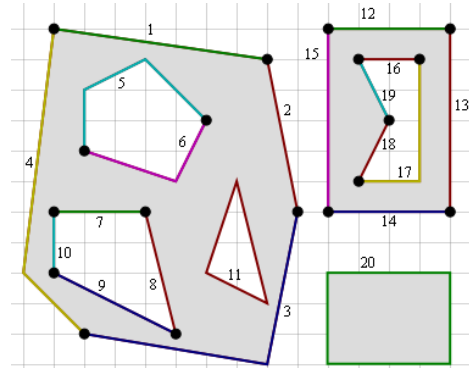


Figure 17: Combined way example 2



## IMPLEMENTATION INITIAL IMPORT

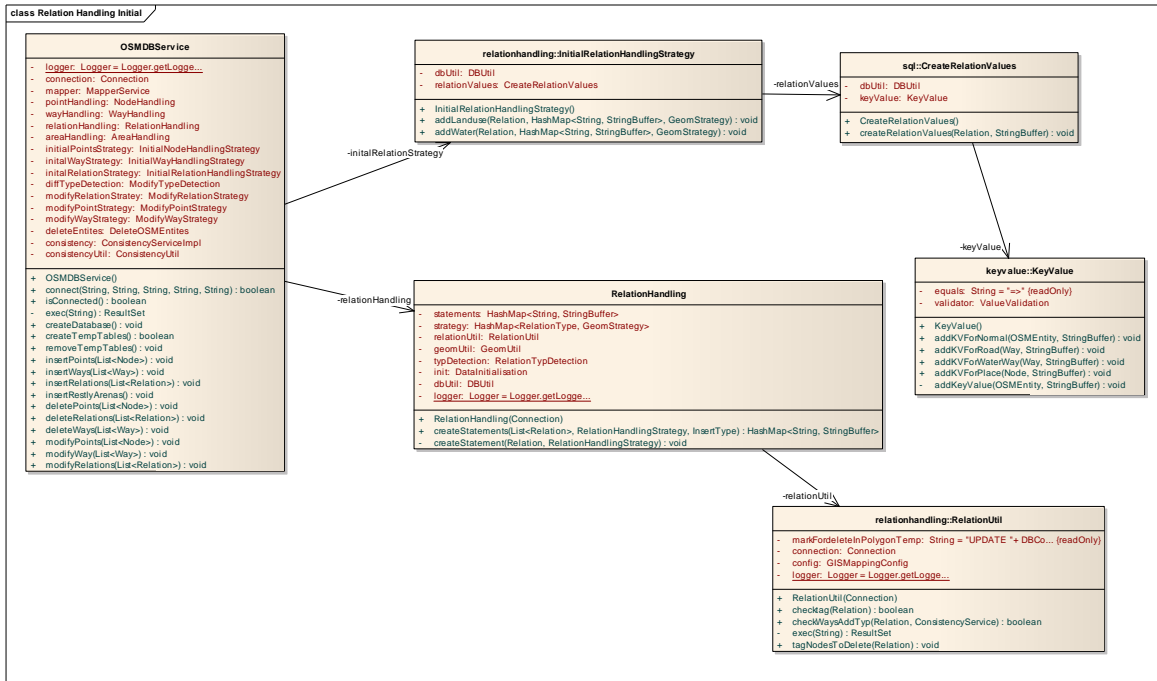


Diagram 28: Class diagram initial relation import

The class diagram shows the needed classes for an initial import implementation.

## IMPLEMENTATION INITIAL IMPORT

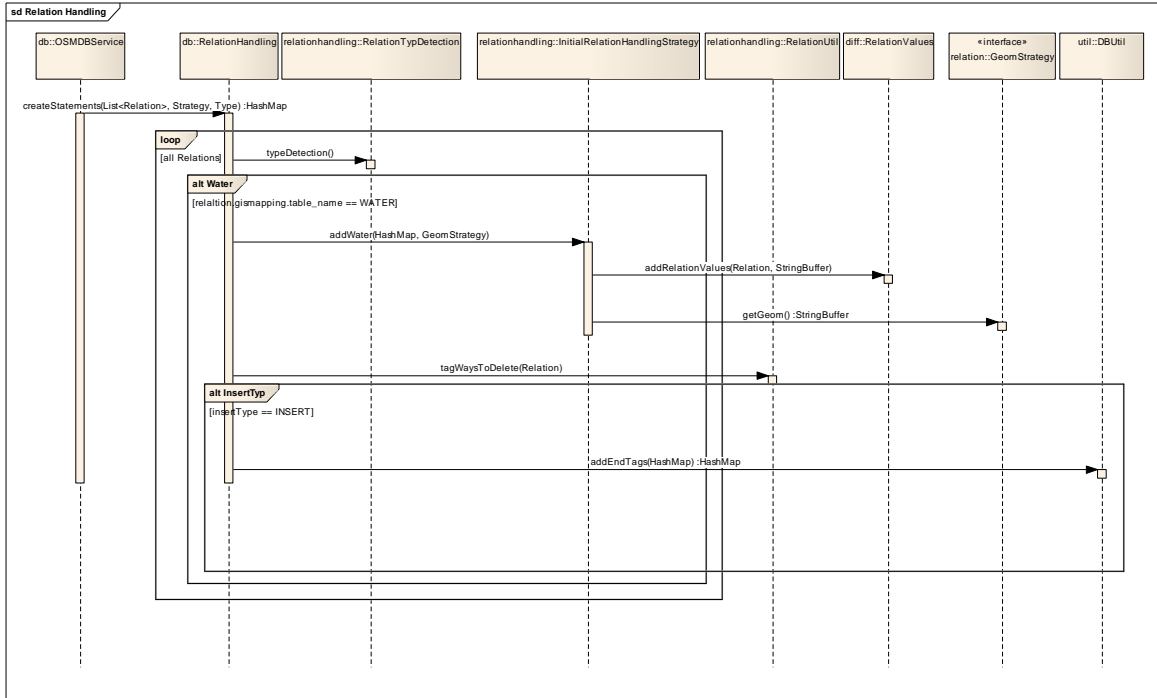


Diagram 29: Sequence diagram initial relation import

### Explanation

1. OSMDService is called from the importer package.
2. OSMDService calls createStatement on RelationHandling and passes the list of relations and the RelationHandlingStrategy for initial import.
  - a. The RelationHandling class firstly detects the type of the relation by calling the detectType method on RelationTypeDetection, the combined ways are also being removed.
  - b. Afterwards, the class detects in which database table the relation has to be imported. In our case this is the table 'water'.
  - c. So RelationHandling calls the addWater(..) method on the InitialRelationHandlingStrategy and passes the GeomStrategy that fits to the relation type detected previously.
3. InitialRelationHandlingStrategy initializes the SQL statement if needed and adds the relation values.
  - a. Afterwards, the method getGeom(..) is called on the GeomStrategy passed to this method.
4. GeomStrategy creates the geom.
5. After the geom has been added to the SQL statement we return to the RelationHandling class.
  - a. In this class we call relationUtil.tagWaysToDelete(Relation) which tags all ways that were used in the relation to be deleted before the areas are imported. This is necessary to prevent a way, which is an area, being imported twice. Once in a relation and once as an area.
6. Add semicolon if needed.

## IMPLEMENTATION DIFFERENTIAL UPDATE

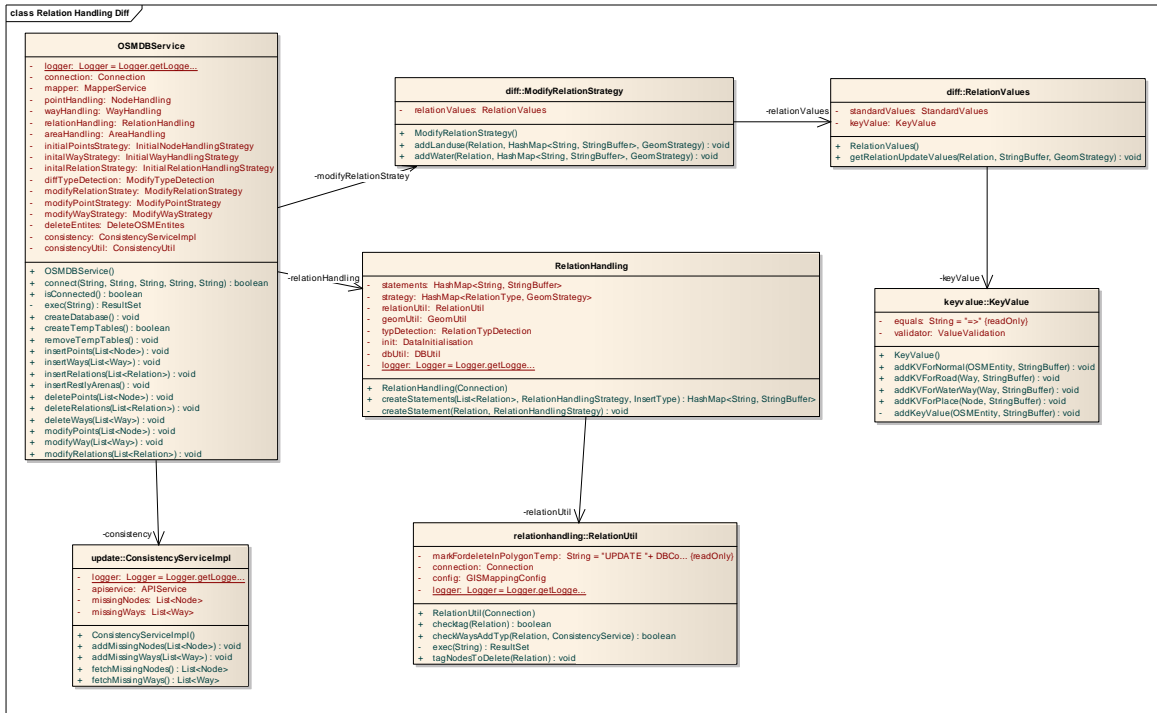


Diagram 30: Class diagram differential relation update

### Note

- The differential update on relations works similar to all the other differential updates.
- Also in this differential update the ConsistencyService is used to make sure that all ways and their nodes are in the expected temporary database tables. If they are not, they will be received by the Consistency Service and inserted into the database.
- TypeDetection works like on initial Import. See 5.3.9

### 5.3.10.3 AREA HANDLING

#### FUNCTIONALITY

Scrolls over the whole polygon\_temp table and searches for areas that we support and create SQL Statements to insert into the database.

#### INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method handleRestlyAreas()
ch.hsr.osminabox.db.sql	Creation of area values
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization, AreaHandlingUtil
ch.hsr.osminabox.db.keyvalue	Helperclass for Key/Value array in database

## IMPLEMENTATION

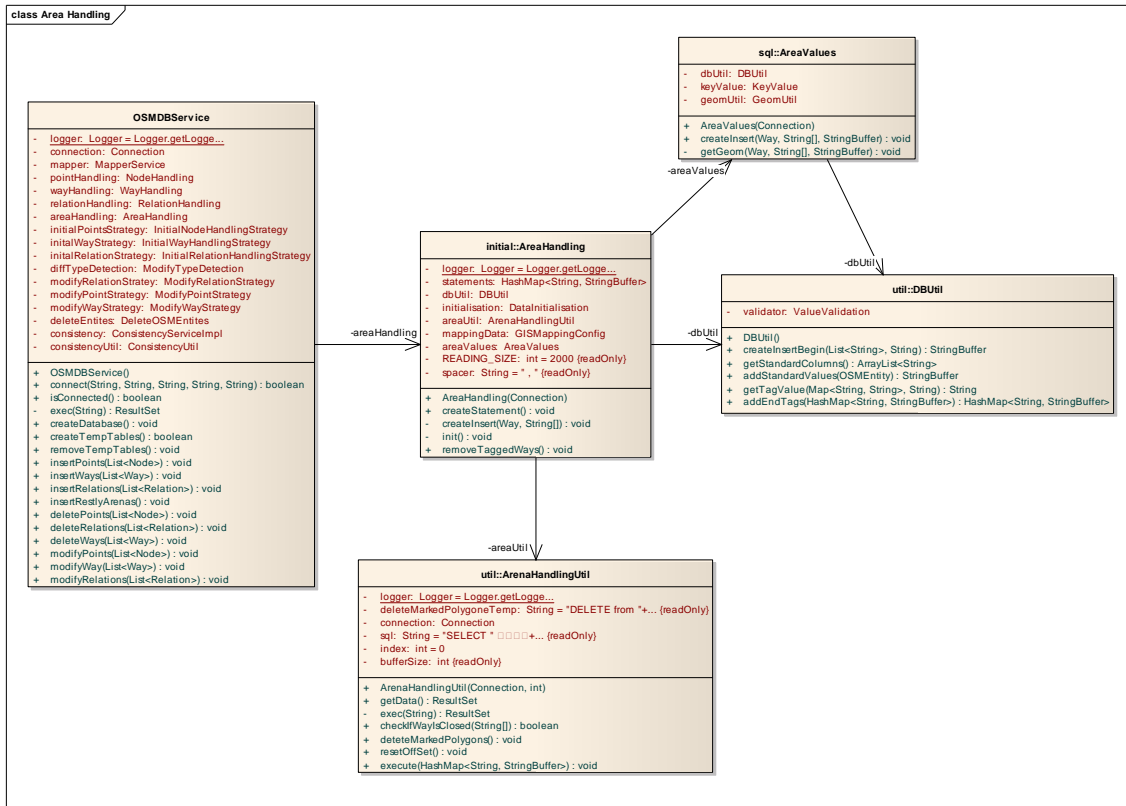


Diagram 31: Class diagram area implementation

This implementation does not need a differential update mechanism, because the differential updates on areas are handled within the modification of the ways.

## SEQUENCEDIAGRAM

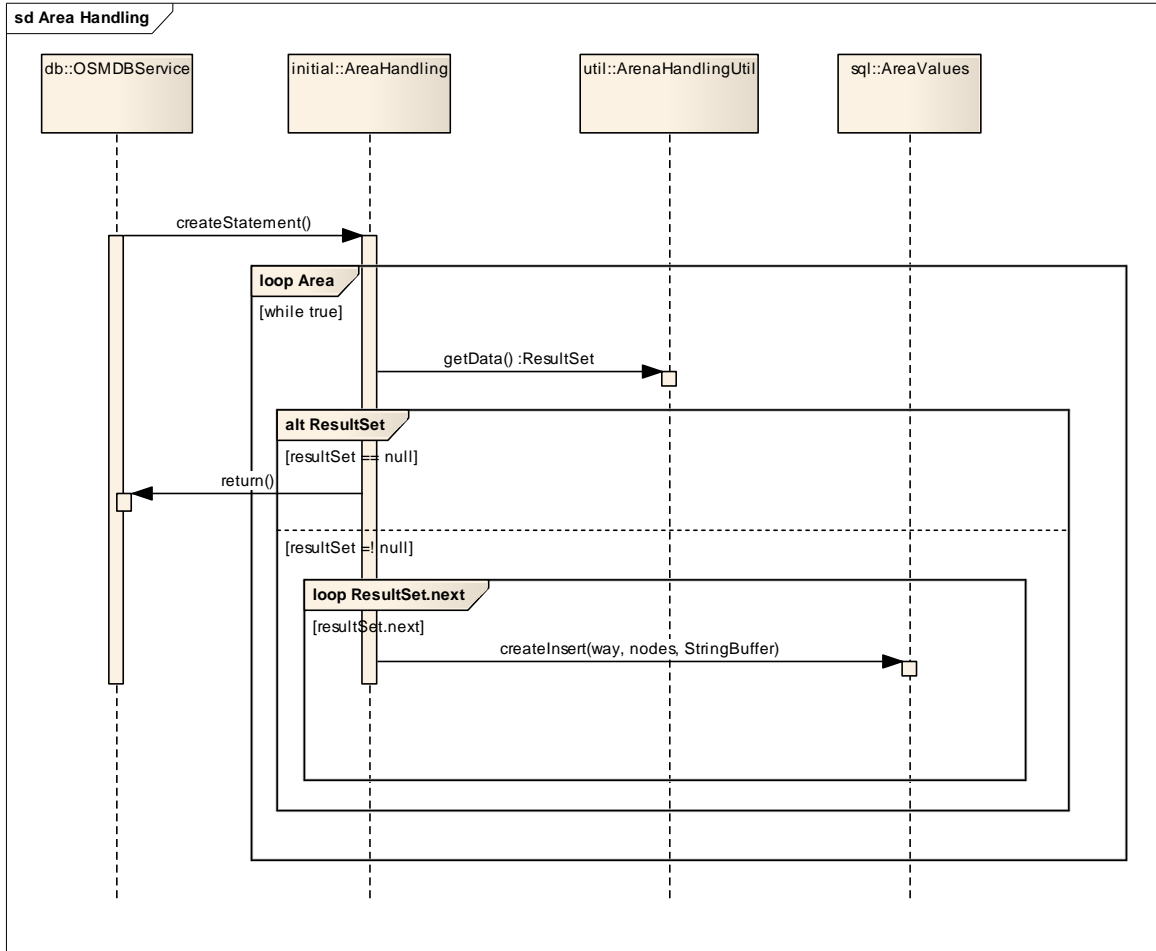


Diagram 32: Sequence diagram area handling

### Explanations

1. After deleting the ways that have been used by a relation, we continue here.
2. OSMDBService calls createStatement on AreaHandling.
3. This call starts a while(true) loop and gets data from AreaUtil.
  - a. AreaUtil returns a ResultSet with the next 2000 entries of the polygon\_temp database table.
4. If the ResultSet is null, we have reached the end of the polygon\_temp table and return.
5. If the ResultSet is not null, we go through every data record and check if we support it. If so we add it to this database table.



### 5.3.11 CONSISTENCY SERVICE

#### FUNCTIONALITY

The ConsistencyService is needed because on an update process it is possible that some references are missing in the update file.

#### INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db.update	Contains the ConsistencyService and some helper classes.

#### IMPLEMENTATION

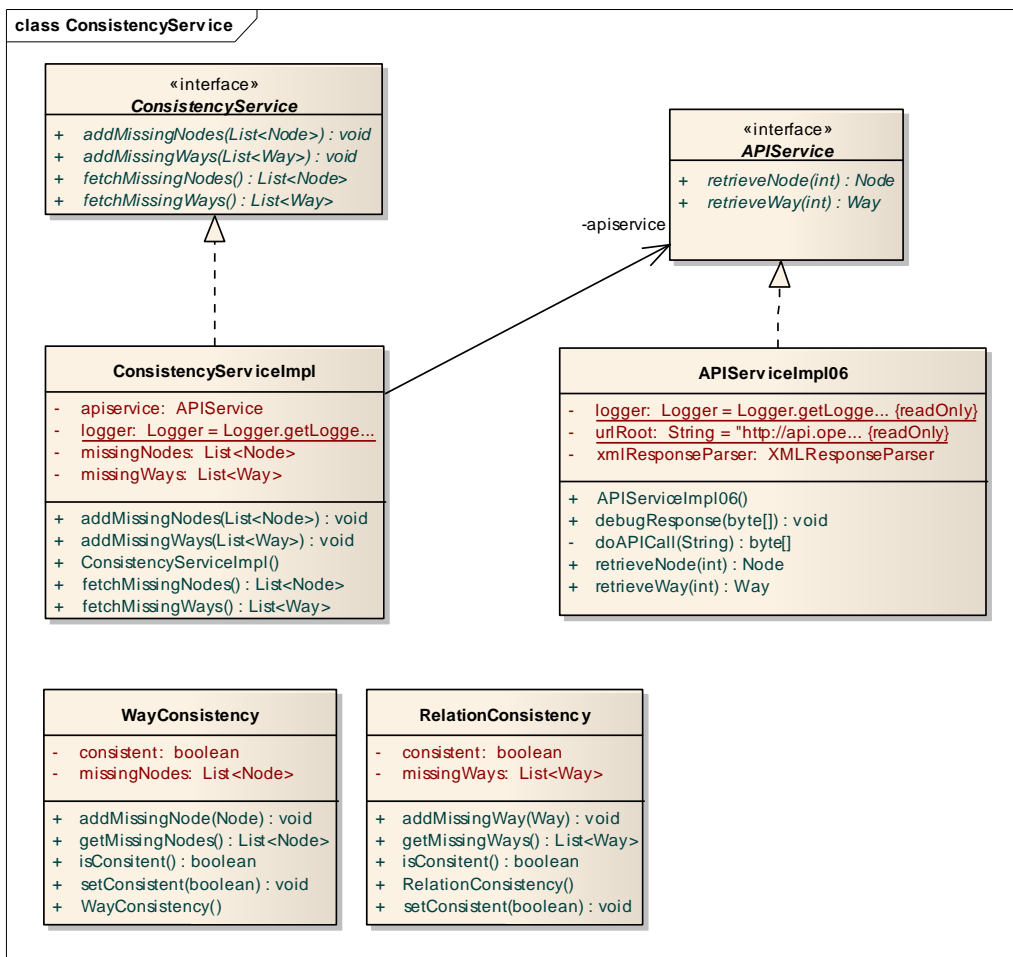


Diagram 33: Class diagram consistency service

The ConsistencyService check the integrity of a way or a relation. It uses the ApiService to fetch missing information. Therefore a connection to the OSM API is always needed if an update process is running.

### 5.3.12 DATA MAPPING

The OSM data is tagged with tags that explain what this node, way or relation represents. A full list of featured tags by OSM can be found here [6d]

For our project we have to filter this big list of OSM tags to a 'normal' set of nodes, ways and relations and map these OSM tags into our database schema. This is done by a file called mapping.xml

Reference for this configuration was a whitepaper from Jochen Topf, "OpenStreetMap Data in Standard GIS Formats" [3].

#### XML SCHEMA

Subsequently you can find a detailed example of the XML file structure.

```

1  <osminabox>
2  <db-schema>
3    <table id="pois">
4      <name>poi</name>
5    </table>
6    ...
7  </db-schema>
8
9  <point-mapping>
10 <mapping>
11 <key>
12 <key-name>place=town</key-name>
13 </key>
14 <ref_table>places</ref_table>
15 <type>town</type>
16 </mapping>
17 ...
18 </point-mapping>
19 <way-mapping>
20 <mapping>
21 <key>
22 <key-name>boundary=administrative</key-name>
23 <key-name>admin_level=1</key-name>
24 </key>
25 <ref_table>boundaries</ref_table>
26 <type>admin_level1</type>
27 </mapping>
28 ...
29 </way-mapping>
30 <polygon-mapping>
31 <mapping>
32 <key>
33 <key-name>landuse=forest</key-name>
34 </key>
35 <ref_table>landuse</ref_table>
36 <type>forest</type>
37 </mapping>
38 ...
39 </polygon-mapping>
40 </osminabox>

```

db-schema: List of all database tables used by OSM-in-a-Box

point-mapping: Mapping information for nodes. An example here could be towns or churches.

way-Mapping: Mapping information for ways. An example here could be roads or railway lines

polygon-mapping: Mapping of areas, for example forest.

## CHANGES FROM WHITEPAPER

Most of the Tags are one-to-one taken over from the whitepaper of Jochen Topf, but there were some inconsistencies inside this whitepaper which led us to add some changes which are listed below.

Chapter	Table	Code	Change
4.1	place	2802	OSM tag is now: highway=services
4.1	place	2803	type: bus_station ; OSM tag: amenity=bus_station
4.1	place	-	NEW added: type: bus_stop ; OSM tag: highway=bus_stop
4.2	poi	2001	type: police ; OSM tag: amenity=police
4.3	pofw	3000	type: place_of_worship
5.1	boundary	1101	type: admin_level1
5.1	boundary	1103	type: admin_level3
5.1	boundary	1104	type: admin_level4
5.1	boundary	1105	type: admin_level5
5.1	boundary	1106	type: admin_level6
5.1	boundary	1107	type: admin_level7
5.1	boundary	1108	type: admin_level8
5.1	boundary	1109	type: admin_level9
5.1	boundary	1110	type: admin_level10
5.2	road	-	new added: type: road; ref_table = roads ; OSM tag: highway=road
5.2	road	-	new added: type: path; ref_table = roads; OSM tag: highway=path
5.2	road	-	new added: type: unsurfaced; ref_table=roads; OSM tag: highway:unsurfaced
5.2	road	-	new added: type: byway; ref_table=roads; OSM tag: highway:byway
5.4	railway	-	new added: type: monorail; ref_table=railways; OSM tag: railway=monorail
5.4	railway	-	new added: type preserved; ref_table=railways; OSM tag: railway=preserved
6.1	landuse	7206	OSM tag: landuse=cemetery
6.1	landuse	7303	type: residential_area
6.1	landuse	-	new added: type: forest; ref_table=landuse; OSM tag: natural=wood
6.1	water	8200	type: water

## TECHNICAL IMPLEMENTATION

The reading of this xml file is done by JAXB [8]. JAXB, an xml library, provides the functionality to create classes out of an xml schema and use them as normal entity classes.

### 5.3.12.1 MAPPER SERVICE

#### FUNCTIONALITY

The MapperService provides the functionality to map an OSMEntity to a corresponding database table according to its tags and entity type. It uses the mapping.xml file described in the chapters above to look up the mappings.

#### INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db.mapper	Contains the MappingService and some helper classes.

#### IMPLEMENTATION

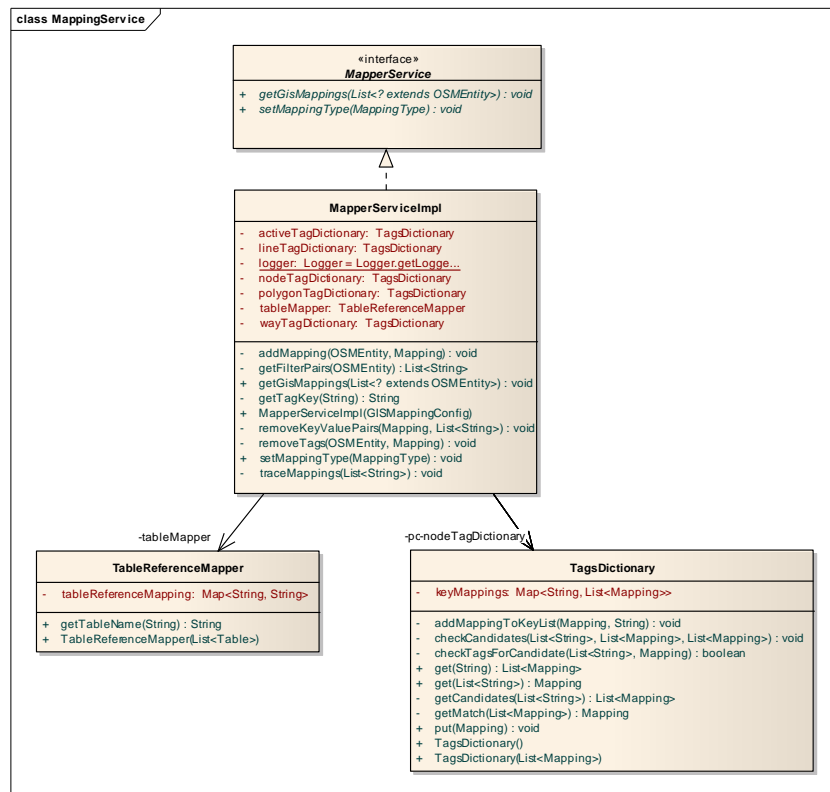


Diagram 34: Class diagram mapping service

#### Explanation:

The MapperServiceImpl class is an implementation of the MapperService. It provides the functionality to map an entity to a table in the database. It contains several tags dictionary, which holds the information about which OSM tag combination leads to a specific table. The TableReferenceMapper provides the table name for a specific table id. The table id is what the TagDictionary returns if a specific mapping has been found.

The MapperService identifies all mappings, since an OSM node can represent more than one entity on a map, and sets the in the OSMEntity class.

### 5.3.13 DATABASE

This project is based on PostgreSQL. Due to the fact that this application has to persistently store GIS data we also use the spatial add-on 'PostGIS'.

#### DATABASE SCHEMA

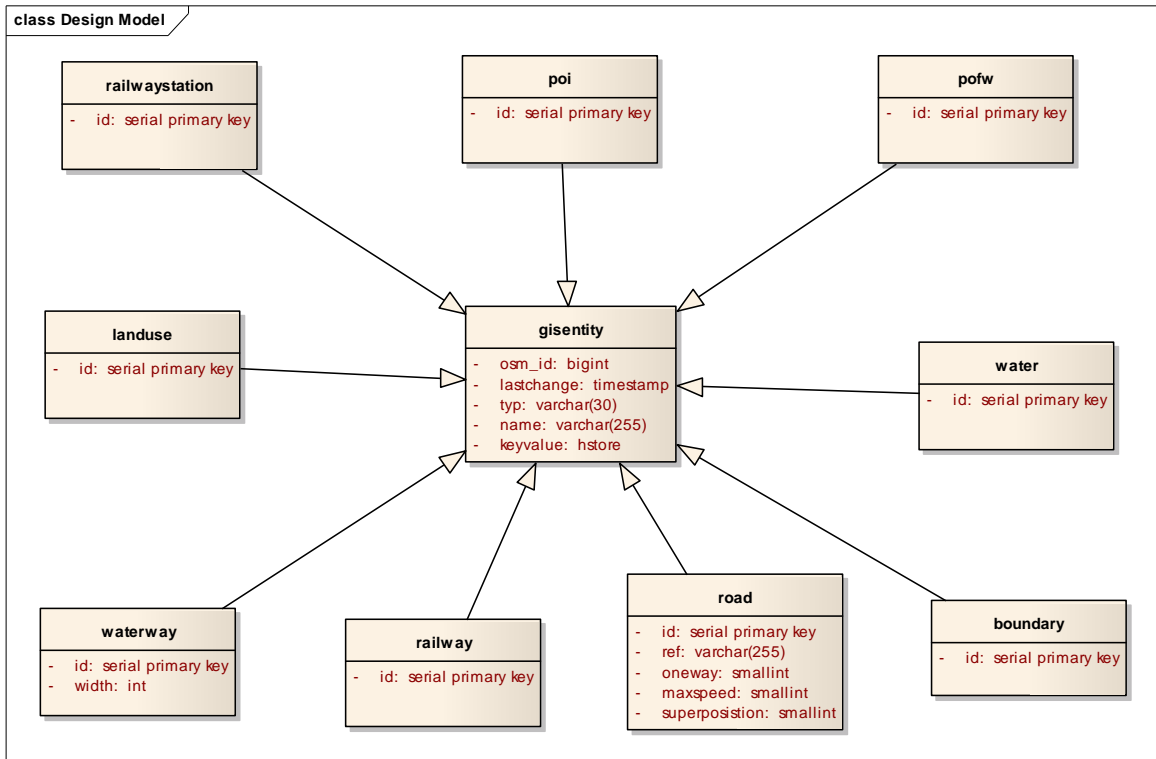


Diagram 35: Database schema

This diagram shows the database schema of the application.

All entities are a generalization of gisentity. The id has to be set in every entity itself, otherwise the serial id would be unique over the whole database. This is not necessary here.

The attribute for the geometrical information is added during the creation process of the database tables by a PostGIS procedure.

#### DATABASE SOFTWARE

The following database software should be used in the correct version. See also installation manual.

Software	Version	Link
PostgreSQL	8.3	<a href="http://www.postgresql.org/">http://www.postgresql.org/</a>
PostGis	1.3.5	<a href="http://postgis.refrains.net/">http://postgis.refrains.net/</a>

### 5.3.14 HOW TO'S

In this chapter we would like to take the time to show what has to be done, if some new functionality should be added to the software.

#### ADD NEW OSM TAG

Adding a new OSM tag is pretty simple.

- Add a new entry in the mapping.xml file with a new type in the according xml-element for point, way or polygon mapping.
- **Attention:** The type of the new entry has to be unique within the whole mapping.xml file!
- Another problem could be that the OSM tag does not fit into any of the existing tables. This case is described in the next chapter below.

#### ADD NEW DATABASE TABLE

Sometimes it can be necessary to add a new database table. For example: If you want to add a new OSM tag but this tag does not fit into any existing table.

This short manual explains what has to be done if you want to add a new database table.

- 1) Add the new table into the config/create\_osm.sql file. This creates the database table on `-t` argument start.
- 2) Now we have to make some decisions
  - a) *If the new table has only standard values*
    - i) Go to step 3.
  - b) *If the new table has the standard values and more*
    - i) Choose what type of OSMEntity it should be (node, way or relation) and add a new 'else if' in the according classes
      - (1) Node: `ch.hsr.osminabox.db.NodeHandling`
      - (2) Way: `ch.hsr.osminabox.db.WayHandling`
      - (3) Relation: `ch.hsr.osminabox.db.RelationHandling`
    - ii) Afterwards follow the flow of the other non-standard values and add your methods in the according classes. See also chapter 'Database layer' in the 'Design' part of this document.
- 3) Add your new entities to mapping.xml in the according xml-element. See also the chapter above.

## 5.4 PROJEKTMANAGEMENT

### 5.4.1 PROJEKTORGANISATION

Das Projektteam für die Bachelorarbeit besteht aus Fabio Renggli, Roland Hof und Michael Huber. Fabio Renggli übernimmt für die Dauer des Projekts die Rolle des Projektleiters. Das Projekt erlaubt es die Verantwortlichkeiten grob auf die drei Projektmitglieder zu verteilen. Diese Aufteilung ist unter Punkt ‚3.1 Organisationsstruktur‘ beschrieben.

Hier ist noch anzufügen, dass die Aufteilung nicht fix ist. Änderungen dieser Aufteilung können während dem Projekt vorgenommen werden.

#### ORGANISATIONSSTRUKTUR

Name	E-Mail	Verantwortlichkeit
<b>Michael Huber (mhuber)</b>	m2huber@hsr.ch	Datenbank und Datenbanksynchronisation
<b>Fabio Renggli (frenggli)</b>	frenggli@hsr.ch	Projektleitung, Infrastruktur, Geoserver
<b>Roland Hof (rhof)</b>	rhof@hsr.ch	XML-Parsing, Scheduling

#### BETREUUNG

Name	Email	Funktion
<b>Stefan Keller</b>	sfkeller@hsr.ch	Betreuung

### 5.4.2 MANAGEMENTABLÄUFE

Für die gesamte Bachelorarbeit stehen 17 Wochen zur Verfügung. Der Projektplan wurde so ausgelegt, dass die 17 Wochen vollständig ausgenutzt werden. Die als Reserve eing geplante Zeit in Woche 16 des Projektes wird entweder als Puffer für allfällige Verzögerungen, oder für die Implementation weiterer Features genutzt.

#### ITERATIONSPLANUNG

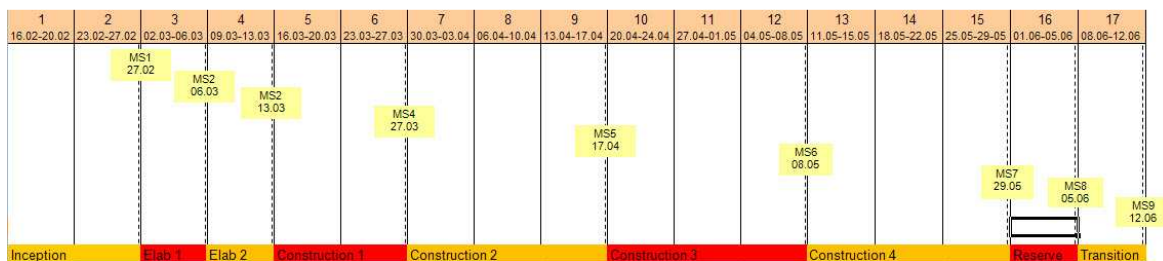


Diagramm 36: Iterationsplanung

### 5.4.2.1 MEILENSTEINE

<b>Meilenstein 1: Projektplan, Anforderungen, Use Case, Technikstudium</b>	27. Februar
<ul style="list-style-type: none"> <li>• Projektplan liegt in einer ersten Version vor.</li> <li>• Anforderungen mit Use Cases sind in einer ersten Version definiert.</li> <li>• Technikstudium ist zum grössten Teil abgeschlossen.</li> </ul>	
<b>Artefakte:</b> -Projektplan -Zeitplan -Anforderungsspezifikation	

<b>Meilenstein 2: Requirements, DomainModell, Evaluation</b>	6. März
<ul style="list-style-type: none"> <li>• Projektplan ist für die nächste Iteration weitergeführt.</li> <li>• Anforderungen müssen überarbeitet sein.</li> <li>• Domainmodell liegt in der ersten Version vor.</li> </ul>	
<b>Artefakte:</b> -Projektplan -Zeitplan -Anforderungen -Domainmodell	

<b>Meilenstein 3: DomainModell, Softwarearchitektur, Arbeitsplatzinstallation</b>	13. März
<ul style="list-style-type: none"> <li>• Projektplan ist für die nächste Iteration weitergeführt.</li> <li>• Domainmodell ist überarbeitet.</li> <li>• Softwarearchitektur ist in erster Version vorhanden.</li> <li>• Der Arbeitsplatz ist soweit installiert, dass mit der Implementation begonnen werden kann.</li> </ul>	
<b>Artefakte:</b> -Projektplan -Domainmodell -Softwarearchitektur	

<b>Meilenstein 4: Softwarearchitektur, Prototyp 1</b>	27. März
<ul style="list-style-type: none"> <li>• Softwarearchitektur ist überarbeitet.</li> <li>• Prototyp 1 ist fertig.</li> <li>• Datenbank: Nodehandling fertig.</li> <li>• Importer: Nodehandling fertig.</li> <li>• Server ist installiert.</li> </ul>	
<b>Artefakte:</b> -Softwarearchitektur -Prototyp 1	



<b>Meilenstein 5: Softwarearchitektur, Prototyp 2</b>	17. April
<ul style="list-style-type: none"> <li>• Softwarearchitektur ist überarbeitet.</li> <li>• Prototyp 2 ist fertig.               <ul style="list-style-type: none"> <li>○ Datenbank: Wayhandling fertig.</li> <li>○ Importer: Wayhandling fertig</li> </ul> </li> <li>• Geoserver konfiguriert</li> </ul>	
<p><b>Artefakte:</b>            -Softwarearchitektur            -Prototyp 2</p>	

<b>Meilenstein 6: Softwarearchitektur, Prototyp 3</b>	8. Mai
<ul style="list-style-type: none"> <li>• Softwarearchitektur ist überarbeitet.</li> <li>• Prototyp 3 ist fertig.               <ul style="list-style-type: none"> <li>○ Datenbank: Relationhandling fertig.</li> <li>○ Importer: Relationhandling fertig</li> </ul> </li> <li>• SLD's erstellt</li> </ul>	
<p><b>Artefakte:</b>            -Softwarearchitektur            -Prototyp 3</p>	

<b>Meilenstein 7: Softwarearchitektur, Prototyp 4</b>	29. Mai
<ul style="list-style-type: none"> <li>• Softwarearchitektur ist überarbeitet.</li> <li>• Prototyp 4 ist fertig.               <ul style="list-style-type: none"> <li>○ Datenbank: Differentialhandling, Boundingbox, Performance-Tuning.</li> <li>○ Importer: Updatehandling, Scheduler.</li> </ul> </li> <li>• Webseite ist fertig.</li> </ul>	
<p><b>Artefakte:</b>            -Softwarearchitektur            -Prototyp 3</p>	

<b>Meilenstein 8: Implementation fertig</b>	5. Juni
<ul style="list-style-type: none"> <li>• Die Programmierarbeiten sind abgeschlossen.</li> <li>• Installer ist fertig.</li> <li>• Softwarearchitekturdokument ist fertig gestellt.</li> </ul>	
<p><b>Artefakte:</b>            -Softwarearchitektur            -Fertiges Produkt</p>	

Meilenstein 9: Transition	12.Juni
<ul style="list-style-type: none"><li>• Abschluss des gesamten Projekts.</li></ul>	
<p><b>Artefakte:</b></p> <ul style="list-style-type: none"><li>-Management Summary</li><li>-Technischer Bericht</li><li>-Erfahrungsberichte</li><li>-Präsentation</li><li>-Poster</li><li>-Projektwiki</li></ul>	

### 5.4.3 RISIKOMANAGEMENT

Risiko	Auswirkung	Schaden in h	Wahrscheinlichkeit in %	Gewichteter Schaden in h	Massnahmen	Restrisiko	Priorität
<b>Datenverlust</b>	Projektstillstand	1-1080	10	0.1 - 180	Einsatz von SVN, Speicherung auf 6 verschiedenen PC's	1%, 18 h	Niedrig
<b>Ausfall von Mitarbeiter</b>	Zeitverschiebung in der Planung	~16	20	3.25	Umstrukturierung des Zeitplans, Rücksprache mit Betreuer, Anforderungen streichen	15%, 2h	Mittel
<b>Unstimmigkeiten im Team</b>	Produktivität sinkt	~30	2	0.6	Gespräch mit Vermittlungsperson / Betreuer	5%, 0.03h	Sehr Niedrig
<b>Definition der zu implementierenden Arbeitspakete nicht möglich</b>	Verzögerung der Implementationsphase	~120h	30	36	Know-How-Aufbau durch Technikstudium, Brainstorming im Projektteam	10%, 3.6h	Mittel
<b>Importer komplexer zu realisieren.</b>	Verzögerung der Implementation	~60h	25	15	Reserve-Iteration wird für die Implementierung verwendet	10%, 1.5h	Mittel
<b>Auftraggeber ändert Meinung über zu implementierende Features</b>	Planung und Priorisierung muss geändert werden.	20-120h	5	1-6 h	Wöchentliche Sitzung, damit Änderungen früh genug aufgenommen werden können. Featurestop definieren.	1%, 0.6h	Niedrig
<b>Genutzte Tool / Libraries sind zu wenig oder schlecht Dokumentiert.</b>	Es wird mehr Zeit gebraucht für die Implementierung.	50 – 300	20	10 – 60 h	Technikstudium durchführen. Mailinglisten oder Beispiele im Internet zur Hilfe nehmen.	10%, 1-6h	Mittel

## 5.4.4 ARBEITSPAKETE

### 5.4.4.1 PROJEKTMANAGEMENT

6.1.1 Projektplan Excel		16 Std
Beschreibung	Das Excel für den Projektplan erstellen und für die Verwendung vorbereiten. Später wird immer für die nächste Iteration vorgeplant.	
Verantwortlich	frenggli	
Abhängigkeiten	-	

6.1.2 Projektplan Dokument		16 Std
Beschreibung	Erstellen des Word Dokument Projektplan. Für jede Iteration wird der Projektplan analog zum Exceldokument erweitert.	
Verantwortlich	frenggli	
Abhängigkeiten	-	

6.1.3 Projektplan Excel Zeiterfassung		8.5 Std
Beschreibung	Nachführen des Excel Sheets gemäss der Zeiterfassung der Teammitglieder.	
Verantwortlich	frenggli	
Abhängigkeiten	6.1.1 Projektplan Excel	

6.1.4 Review Projektplan		6 Std
Beschreibung	Review des Projektplans Excel und Word.	
Verantwortlich	mhuber / rhof	
Abhängigkeiten	6.1.1 Projektplan Excel 6.1.2 Projektplan Dokument	

### 5.4.4.2 REQUIREMENTS

6.2.1 Anforderungsspezifikation		10 Std
Beschreibung	Erstellen der Anforderungsspezifikation mit funktionalen und nicht funktionalen Anforderungen, sowie integration der Use Cases.	
Verantwortlich	rhof	
Abhängigkeiten	6.2.2 Use Cases	

6.2.3 Review Anforderungsspezifikation		4 Std
Beschreibung	Review der Anforderungsspezifikation.	
Verantwortlich	frenggli	
Abhängigkeiten	6.2.1 Anforderungsspezifikation	

### 5.4.4.3 ANALYSE

6.3.1 Domainanalyse		6 Std
Beschreibung	Dokument für die Domainanalyse schreiben. Domainmodell mit dem Enterprise Architect erstellen und im Dokument integrieren.	
Verantwortlich	mhuber	
Abhängigkeiten	-	

6.3.3 SSD + Contracts		2.5 Std
Beschreibung	Systemsequenzdiagramme und Contracts für Domainanalyse erstellen.	
Verantwortlich	mhuber	
Abhängigkeiten	-	

<b>6.3.5 Review Domainanalyse</b>		<b>3 Std</b>
Beschreibung	Domainanalyse wird reviewed	
Verantwortlich	frenggli	
Abhängigkeiten	6.3.1 Domainanalyse 6.3.2 SSD + Contracts	

<b>6.3.6 Evaluation Datenbankstruktur</b>		<b>4 Std</b>
Beschreibung	Es wird eine Evaluation durchgeführt, in welcher herausgefunden werden soll, wie die OSM-Daten in die PostgreSQL-Datenbank abgelegt werden müssen, damit sie für die Verwendung von GeoServer optimiert ist.	
Verantwortlich	frenggli	
Abhängigkeiten	-	

<b>6.3.7 Evaluation WFS</b>		<b>4 Std</b>
Beschreibung	Es wird eine Evaluation durchgeführt, in welcher herausgefunden werden soll, wie die Daten auszusehen haben, welche über die WFS-Schnittstelle zurückgegeben werden.	
Verantwortlich	rhof	
Abhängigkeiten	-	

<b>6.3.8 Definition der Implementationsarbeiten</b>		<b>6 Std</b>
Beschreibung	Es wird durch ein Brainstorming definiert, welche Teile des bestehenden OpenStreetMap-Projektes in dieser Arbeit umgesetzt werden.	
Verantwortlich	frenggli / rhof / mhuber	
Abhängigkeiten	-	

#### 5.4.4.4 DESIGN ANALYSE

<b>6.4.1 Softwarearchitektur Dokument</b>		<b>99 Std</b>
Beschreibung	Die Softwarearchitektur wird in einem Dokument erfasst. Dazu gehören unter anderem die Erklärung der Paketstruktur, der Klassen und aussagekräftige Sequenzdiagramme.	
Verantwortlich	mhuber / rhof	
Abhängigkeiten	-	

<b>6.4.2 Softwarearchitektur Prototyp</b>		<b>15 Std</b>
Beschreibung	Es wird ein Prototyp für die weiteren Arbeiten erstellt und somit die Durchführbarkeit bewiesen.	
Verantwortlich	mhuber / rhof	
Abhängigkeiten	-	

<b>6.4.3 Review Softwarearchitektur</b>		<b>19 Std</b>
Beschreibung	Domainanalyse wird reviewed	
Verantwortlich	frenggli / rhof / mhuber	
Abhängigkeiten	6.3.1 Softwarearchitektur Dokument	

#### 5.4.4.5 IMPLEMENTATION

<b>6.5.1 Automatische Tabellenerstellung</b>		<b>4 Std</b>
Beschreibung	Die Datenstruktur der Datenbank soll der Java-Applikation soll automatisch erstellt werden.	
Verantwortlich	mhuber	
Abhängigkeiten		

<b>6.5.2 Mapping</b>		<b>16 Std</b>
Beschreibung	Das Mapping zwischen OSM-Tags und den Datenbank-Tabellen wird auf Parser- und Datenbankebene implementiert.	
Verantwortlich	mhuber /rhof	
Abhängigkeiten		

<b>6.5.3 Architekturgerüst</b>		<b>15 Std</b>
Beschreibung	Die allgemeine Applikationsstruktur wird erstellt.	
Verantwortlich	rhof	
Abhängigkeiten		

<b>6.5.4 Applikationskontext</b>		<b>10 Std</b>
Beschreibung	Der Applikationskontext wird erstellt.	
Verantwortlich	rhof	
Abhängigkeiten		

<b>6.5.5 Argument Parsing</b>		<b>8 Std</b>
Beschreibung	Das Parsen der Startparameter wird implementiert.	
Verantwortlich	rhof	
Abhängigkeiten		

<b>6.5.6 Node Handling</b>		<b>28 Std</b>
Beschreibung	Es wird implementiert wie die einzelnen Nodes gehandelt werden. Dies geschieht auf Parser- und auf Datenbankebene.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		

<b>6.5.7 Way Handling</b>		<b>58 Std</b>
Beschreibung	Es wird implementiert wie die einzelnen Ways gehandelt werden. Dies geschieht auf Parser- und auf Datenbankebene.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		

<b>6.5.8 Relation Handling</b>		<b>82 Std</b>
Beschreibung	Es wird implementiert wie die einzelnen Relations gehandelt werden. Dies geschieht auf Parser- und auf Datenbankebene.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		

<b>6.5.9 Differential Handling</b>		<b>65 Std</b>
Beschreibung	Es wird implementiert wie das Update durchgeführt wird beim Import der Differential-Files. Dies geschieht auf Parser- und auf Datenbankebene.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		

<b>6.5.10 Scheduler</b>		<b>21 Std</b>
Beschreibung	Der Scheduler wird implementiert.	
Verantwortlich	rhof	
Abhängigkeiten		
<b>6.5.11 Bounding Box</b>		<b>15 Std</b>
Beschreibung	Die Boundingbox-Strategie wird auf Parser- und auf Datenbankebene implementiert.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		
<b>6.5.12 Key- / Value-Einbau</b>		<b>5 Std</b>
Beschreibung	Es wird implementiert, dass alle Tags von OSM in eine spezielle Key- / Value-Spalte in der Datenbank importiert werden.	
Verantwortlich	mhuber	
Abhängigkeiten		
<b>6.5.13 Daten Korrekturen</b>		<b>10 Std</b>
Beschreibung	Importierungsänderungen, welche anhand der visualisierten Map gefunden wurden	
Verantwortlich	mhuber	
Abhängigkeiten		
<b>6.5.14 Performance Verbesserungen</b>		<b>4 Std</b>
Beschreibung	Implementierungsänderung zur Performancesteigerung.	
Verantwortlich	mhuber	
Abhängigkeiten		
<b>6.5.15 Cleanup / Refactoring / Debugging</b>		<b>47 Std</b>
Beschreibung	Es werden Cleanups, Refactorings und Debugging der gesamten Applikation durchgeführt.	
Verantwortlich	rhof / mhuber	
Abhängigkeiten		
<b>6.5.16 Infrastruktur</b>		<b>4 Std</b>
Beschreibung	Installation der Infrastruktur	
Verantwortlich	frenggli	
Abhängigkeiten		
<b>6.5.17 Buildfile</b>		<b>4 Std</b>
Beschreibung	Buildfile zur automatisierten Compilieren, Testen und Packen der Applikation.	
Verantwortlich	frenggli	
Abhängigkeiten		
<b>6.5.18 SLD</b>		<b>74 Std</b>
Beschreibung	Erstellen der Style Layer Deskriptoren	
Verantwortlich	frenggli	
Abhängigkeiten		
<b>6.5.19 Geoserver Konfiguration</b>		<b>43 Std</b>
Beschreibung	Konfigurieren des GeoServer.	
Verantwortlich	frenggli	
Abhängigkeiten	6.5.16 Infrastruktur	

6.5.20 Webseite		25 Std
Beschreibung	Erstellen der Demonstrations-Webseite mit Hilfe von OpenLayers	
Verantwortlich	frenggli	
Abhängigkeiten		

6.5.21 Installer		35 Std
Beschreibung	Installer für Windows und Unix erstellen.	
Verantwortlich	frenggli	
Abhängigkeiten		

#### 5.4.4.6 QUALITÄTSMASSNAHMEN

6.6.1 Technikstudium		75 Std
Beschreibung	Einlesen in die verschiedenen zu verwendenden Technologien.	
Verantwortlich	frenggli / mhuber / rhof	
Abhängigkeiten	-	

6.6.2 Codereview		30 Std
Beschreibung	Review des Codes, welcher während den Iterationen implementiert wurde.	
Verantwortlich	frenggli / mhuber / rhof	
Abhängigkeiten	-	

#### 5.4.4.7 DOKUMENTATION

6.7.1 Abstract		2 Std
Beschreibung	Abstrakt wird geschrieben	
Verantwortlich	frenggli	
Abhängigkeiten		

6.7.2 Broschüre		8 Std
Beschreibung	Broschüre wird anhand der Vorlage erstellt.	
Verantwortlich	rhof	
Abhängigkeiten		

6.7.3 Management Summary		5 Std
Beschreibung	Das Management Summary wird verfasst.	
Verantwortlich	mhuber	
Abhängigkeiten		

6.7.4 Technischer Bericht		10 Std
Beschreibung	Erarbeiten des technischen Berichtes	
Verantwortlich	rhof	
Abhängigkeiten		

6.7.5 Installationsdokument		5 Std
Beschreibung	Die Installationsanleitung wird geschrieben.	
Verantwortlich	frenggli	
Abhängigkeiten	6.5.21 Installer	



<b>6.7.6 Usermanual</b>		<b>10 Std</b>
Beschreibung	Erstellen des Usermanuals	
Verantwortlich	rhof	
Abhängigkeiten		

<b>6.7.7 Serverkonfigurationsdokument</b>		<b>15 Std</b>
Beschreibung	Dokumentieren der Installations- und Konfigurationsarbeiten.	
Verantwortlich	frenggli	
Abhängigkeiten		

<b>6.7.8 Plakat</b>		<b>10 Std</b>
Beschreibung	Erstellen des Plakates	
Verantwortlich	frenggli	
Abhängigkeiten		

<b>6.7.9 Gesamtdokumentation</b>		<b>40 Std</b>
Beschreibung	Alle Dokumente werden abgabefertig gemacht und zu einer Gesamtdokumentation zusammengefügt.	
Verantwortlich	frenggli / mhuber / rhof	
Abhängigkeiten		

#### 5.4.4.8 SITZUNGEN

<b>6.8.1 Sitzungen</b>		<b>76.5 Std</b>
Beschreibung	1.5 h pro Woche	
Verantwortlich	mhuber / frenggli / rhof	
Abhängigkeiten	-	

<b>6.8.2 Sitzungsprotokoll</b>		<b>8.5 Std</b>
Beschreibung	Sitzungsprotokoll erstellen.	
Verantwortlich	mhuber / frenggli / rhof	
Abhängigkeiten	6.9.1 Sitzungen	

<b>6.8.3 Sitzungsprotokoll Review</b>		<b>5.1 Std</b>
Beschreibung	Review des Sitzungsprotokolls.	
Verantwortlich	mhuber / frenggli / rhof	
Abhängigkeiten	6.9.2 Sitzungsprotokoll	

---

## 5.4.5 INFRASTRUKTUR

Als Infrastruktur dienen uns folgende Komponenten:

- Linux Server:
  - <http://sinv-56018.edu.hsr.ch>
    - Tomcat 5.5, Port 8080
    - PostgreSQL, Port 5432
- Ein Developer-Wiki:
  - <http://dev.ifs.hsr.ch/osminabox/>
- SVN-Repository:
  - <http://sifsv002.hsr.ch/svn/osminabox/>
- Daily-Build mit Ant.

## ENTWICKLUNGSUMGEBUNG

Als Entwicklungsumgebung werden wir die einzelnen Laptops der Teammitglieder sowie die Arbeitsplatz-Computer der HSR verwenden. Somit verfügt jeder über seine eigene Arbeitsumgebung.

Der Code wird zentral im SVN revisioniert.

## ENTWICKLUNGSSOFTWARE VERSION

Die Entwicklung wird mit folgender Software durchgeführt:

- Eclipse Ganymede
- Java JDK 1.6
- PostgreSQL 8.3
- PostGIS 1.3.5
- Apache Tomcat 5
- Geoserver 1.7.2

---

## 5.4.6 QUALITÄTSSICHERUNG

### CODE RICHTLINIEN

- Die von uns verwendeten Code Richtlinien basieren auf den Standardeinstellungen des Code Formatters von Eclipse.
- Die Funktionsnamen müssen aussagekräftig gewählt werden. Die Namen sollten möglichst ohne Kommentare auf den Zweck der Methode deuten.
- Der Inhalt komplexer Funktionen wird mithilfe von Javadoc beschrieben.
- Der Code innerhalb einer Methode wird sauber strukturiert.
- Es sollen die in den Software Engineering gelernten Praktiken und Standards angewandt werden. (Patterns, Code Richtlinien)

## REVIEWS

Wir führen kontinuierlich Reviews durch. In diesen werden die Coderichtlinien kontrolliert und geprüft ob die Anwendung den Anforderungsspezifikationen entspricht.

Die Reviews sind als Arbeitspakete erfasst und werden in den zugehörigen Iterationen in den Arbeitsaufwand einberechnet.

## TESTPLANUNG

Während der Entwicklung werden die einzelnen Programmteile unter Verwendung von Unit-Tests einer Prüfung unterzogen. Des Weiteren wird in den Code Reviews nach Fehlern gesucht damit diese behoben werden können. Für die Erstellung der jeweiligen Testsznarien ist der Entwickler des entsprechenden Programmteils selbst verantwortlich.

Nach Abschluss der Implementationsphasen werden funktionale Tests am Endprodukt durchgeführt, um deren Qualität auf messbare Werte zu bringen.

## 5.5 PROJEKTMONITORING

### 5.5.1 SOLL-IST-ZEIT-VERGLEICH

Anschließend eine Tabelle mit den Ist- und Soll-Zeiten. Die genauen Angaben zu den einzelnen Paketen innerhalb der Überpaketen können dem Excel Zeitplan entnommen werden.

Überpakete	Soll	Ist	Differenz
Projekt Management	46.5	43.0	3.5
Requirements	18.0	15.5	2.5
Analyse	25.5	18.0	0.5
Design Analyse	133.0	68.0	65.0
Implementation	573.0	637.5	-64.5
Qualitätsmassnahmen	105.0	94.0	11.0
Dokumentation	92.0	95.0	-3.0
Sitzungen	85.0	71.9	13.1

### 5.5.2 CODESTATISTIK

Die folgenden Codestatistiken wurden mit Structure101 generiert:

Size	
<b>Jars (and / or classpath directories)</b>	1
<b>Packages (that contain classes)</b>	25
<b>Classes</b>	152
<b>NI(Number of bytecode Instructions)</b>	15'000
<b>LOC(Non Comment Non Blank Lines of Code)</b>	6'000

Term	Threshold	#Offenders	Offenses (%)	XS contribution
<b>Tangled (design)</b>	0	0 of 5	0%	0%
<b>Fat (design)</b>	120	0 of 5	0%	0%
<b>Fat (leaf package)</b>	120	0 of 25	0%	0%
<b>Fat (class)</b>	120	0 of 152	0%	0%
<b>Fat (method)</b>	15	1 of 693	0%	100%
<b>Total</b>		1	0%	

### 5.5.3 SITZUNGSPROTOKOLLE

Es wird darauf verzichtet, die Sitzungsprotokolle hier einzufügen. Die Sitzungsprotokolle sind auf der CD enthalten.

## 5.6 SOFTWAREDOKUMENTATION

### 5.6.1 SERVER KONFIGURATION

#### 5.6.1.1 INSTALLATION

Als Grundinstallation wurde eine CentOS Version 5 verwendet. Es wurde nur das absolute Minimum von der DVD installiert, damit im Nachhinein mit Hilfe von *yum* die gewünschten Packages installiert werden konnten. Damit wurde erreicht, dass ein massgeschneidertes System installiert wurde, welches über keine überflüssigen und verwirrenden Komponenten verfügt.

#### SERVERINFORMATIONEN

Für diese Bachelorarbeit standen zwei Server zur Verfügung. Wobei anzumerken ist, dass der zweite Server ein Klon des ersten ist und verwendet wurde um den Installer zu testen.

Auf Server **sinv-56018.edu.hsr.ch** stehen alle Funktionen von OSM-in-a-Box bereits zur Verfügung.

Auf Server **sinv-56028.edu.hsr.ch** ist die Grundinstallation vorhanden, jedoch wurde OSM-in-a-Box noch nicht installiert.

Parameter	Wert
Name	<b>sinv-56018.edu.hsr.ch / sinv-56028.edu.hsr.ch</b>
IP	152.96.56.18 / 152.96.56.28
OS	CentOS 5
Passwort	HSR-773399

#### PAKET INSTALLATION

Es wurden folgende Pakete mit yum installiert:

- Java 1.6
- Tomcat 5
- PostgreSQL 8.3
- PostGIS 1.3.5
- Apache 2.2.3
  - mod\_proxy\_ajp

Zusätzlich wurde die Webapplikation GeoServer im Tomcat deployed.

### 5.6.1.2 KONFIGURATION

Standardinstallationen, die nicht speziell konfiguriert werden müssen, sind hier nicht erwähnt. Bei den Konfigurationsdateien ist der Fokus auf Änderungen der Standardinstallationen gelegt.

#### TOMCAT5

Bei der Tomcat-Installation muss die Datei `/etc/tomcat5/tomcat5.conf` folgendermassen angepasst werden.

```
CATALINA_OPTS="-Xms64m -Xmx1024m -server -XX:PermSize=32 -XX:MaxPermSize=128m"
```

Erklärung der verschiedenen Argumente:

Switch	Erklärung
<b>-Xms64m</b>	Definiert wie gross der Heap beim Start der Java Virtual Maschine (JVM) sein soll. Das <i>m</i> am Ende gibt an das der Wert in Megabyte ist.
<b>-Xmx1024m</b>	Definiert wie gross der Heap der JVM maximal anwachsen darf.
<b>-server</b>	Teilt der JVM mit, dass sie im Modus mit höherer Performance laufen soll. Die Speichereffizienz sinkt deshalb jedoch.
<b>-XX:PermSize=32</b>	Definiert den Speicher der automatisch für die JVM reserviert wird.
<b>-XX:MaxPermSize=128m</b>	Definiert den maximalen Speicher der automatisch für die JVM reserviert wird.

#### APACHE 2.2.3

Es ist wichtig das bei der Installation von Apache, dass das Modul `mod_proxy_ajp` mit installiert wird. Dieses Modul wird als Proxy verwendet, welcher von den Tomcat geschaltet wird. Bei einer Standardinstallation mit yum ist dieses Modul bereits vorhanden. Um den Proxy zu konfigurieren müssen folgende Anpassungen in den Konfigurationsdateien gemacht werden.

`/etc/httpd/conf.d/proxy_ajp.conf`

```
ProxyPass /geoserver/ ajp://localhost:8009/geoserver/
ProxyPassReverse /geoserver/ ajp://localhost:8009/geoserver/
```

Durch diese Änderungen wird Apache mitgeteilt, dass bei einem Aufruf auf `/geoserver/` der Aufruf direkt via ajp an Tomcat weitergeleitet wird.

#### POSTGRESSQL 8.3

Um PostgreSQL richtig zu konfigurieren muss folgende Konfigurationsdateien angepasst werden.

- `/var/lib/pgsql/pg_hba.conf`
- `/var/lib/pgsql/postgresql.conf`

`Pg_hba.conf`:

```
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 152.96.0.0/16 md5
```

Diese zwei Zeilen legen fest, das Lokal wie auch vom gesamten HSR-Subnet auf die Datenbank zugegriffen werden kann. Damit PostgreSQL auf allen Network-Interfaces angesprochen werden kann, muss noch folgende Zeile im postgresql.conf gesetzt werden:

*postgresql.conf:*

```
# - Connection Settings -  
listen_addresses = '*'      # what IP address(es) to listen on;  
                           # comma-separated list of addresses;  
                           # defaults to 'localhost', '*' = all
```

## POSTGIS

Für die Installation von PostGIS müssen folgende Befehle ausgeführt werden:

```
tar xvfz postgis-1.3.6.tar.gz  
cd postgis-1.3.6  
./configure  
make  
make install
```

## DATENBANKKONFIGURATION

Folgende Befehle werden verwendet, um die für unser Projekt erforderlichen Konfigurationen auszuführen:

```
su - postgres.  
createuser -P osm  
createdb -O osm -U osm -W osm  
exit  
createlang plpgsql -U osm -W  
psql -U osm -W -d osm -f /usr/share/postgresql/contrib/hstore.sql  
psql -U osm -W -d osm -f /usr/share/postgresql/contrib/lwpostgis.sql  
psql -U osm -W -d osm -f /usr/share/postgresql/contrib/spatial_ref_sys.sql
```

Die drei letzten Befehle führen die SQL-Skripte aus, welche benutzt werden um die Datenbank PostGIS- und Hstore tauglich zu machen.

## GEOSEVER

Sofern die Konfigurationen nicht speziell durchgeführt werden müssen, wird in diesem Kapitel nicht weiter darauf hingewiesen, sondern nur erläutern was und zu welchem Zweck konfiguriert wurde. Weitere Informationen sind hier zu finden [5]

## Namespace

Der Namespace wird benötigt um einen eindeutigen URI zu definieren.

<b>Prefix</b>	Osm

## Datastore

Aufbauend auf den Namespace wird der Datastore konfiguriert. Darin werden die Informationen definiert, welche GeoServer benötigt um auf die Datenbank zuzugreifen. Folgende Parameter sind anzupassen:

<b>Host</b>	localhost
<b>Port</b>	5432
<b>Schema</b>	public
<b>Database</b>	osm
<b>User</b>	osm
<b>Passwd</b>	osm

## STYLES

Folgende Styles sind bereits vordefiniert:

- osm\_boundary
- osm\_coastline
- osm\_landuse
- osm\_place
- osm\_pofw
- osm\_poi
- osm\_railway
- osm\_railwaystation
- osm\_road
- osm\_water
- osm\_waterway

Als Vorlage für die Styles wurden die Konfigurationsdateien von Mapnik verwendet. Mapnik verwendet keinen Open Geospatial (OGS) konformen Standard. Somit mussten die Regeln von Mapnik zum OGS konformen Style Layer Descriptor (SLD) konvertiert werden. Da die Konvertierungsschritte nicht eindeutig und wiederholend vorgenommen werden können, wurde entschieden, dass diese von Hand ausgeführt werden. Es wurde somit auf die Implementation eines Konverters verzichtet. Im folgenden Abschnitt wird anhand eines Beispiels das allgemeine Vorgehen beschrieben.



### Mapnik-Regel:

```
<Rule>
  <Filter>[highway] = 'motorway' or [highway] = 'motorway_link'</Filter>
  <MaxScaleDenominator>500000</MaxScaleDenominator>
  <MinScaleDenominator>200000</MinScaleDenominator>
  <LineSymbolizer>
    <CssParameter name="stroke">#809bc0</CssParameter>
    <CssParameter name="stroke-width">2.5</CssParameter>
  </LineSymbolizer>
</Rule>
```

### SLD-Regel:

```
<Rule>
  <Name>Motorway_Link/Motorway max 500000</Name>
  <ogc:Filter>
    <ogc:Or>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>typ</ogc:PropertyName>
        <ogc:Literal>motorway_link</ogc:Literal>
      </ogc:PropertyIsEqualTo>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>typ</ogc:PropertyName>
        <ogc:Literal>motorway</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:Or>
  </ogc:Filter>
  <MaxScaleDenominator>500000</MaxScaleDenominator>
  <MinScaleDenominator>200000</MinScaleDenominator>
  <LineSymbolizer>
    <Stroke>
      <CssParameter name="stroke">#809bc0</CssParameter>
      <CssParameter name="stroke-width">2.5</CssParameter>
    </Stroke>
  </LineSymbolizer>
</Rule>
```

1. Die Regel muss in der Mapnik-Datei gefunden werden.
2. Der Filter muss auf den OGS-Standard angepasst werden.
3. Die Darstellungsparameter müssen umgeschrieben werden.

## Probleme:

Es besteht noch das Problem, dass der GeoServer die definierte Schriftart nicht übernimmt. Dieses Problem konnte trotz Nachfrage auf der GeoServer-Mailingliste nicht gelöst werden.

## FEATURETYPE

Bei GeoServer ist ein FeatureType gleich einem Layer in der späteren Darstellung. Pro Datenbanktabelle wurde ein FeatureType erstellt und konfiguriert. Die nachstehende Liste zeigt die von uns konfigurierten FeatureTypes an. Der gleichnamige Style wurde als Standardstyle definiert.

- osm\_ds:boundary
- osm\_ds:coastline
- osm\_ds:landuse
- osm\_ds:place
- osm\_ds:pofw
- osm\_ds:poi
- osm\_ds:railway
- osm\_ds:railwaystation
- osm\_ds:road
- osm\_ds:water
- osm\_ds:waterway

Es muss folgendes konfiguriert werden.

<b>Bounding box</b>	-180 / -90 – 180 / 90
<b>Caching Enable</b>	false

Die Boundingbox wurde standardmässig so gewählt, damit die ganze Welt repräsentiert wird. Das Caching wurde ausgeschaltet, weil dafür GeoWebCache eingesetzt wird. Die restlichen Konfigurationsparameter wurden vom GeoServer so übernommen.

## WMS

Die allgemeinen Einstellungen vom Web Map Service (WMS) wurden, wie vom GeoServer vorgeschlagen, bei den Standartwerten belassen. Um jedoch die verschiedenen FeatureTypes übereinander anzuzeigen wurden zwei Base Maps erstellt:

- osm
- osm\_no\_boundary

Die beiden Base Maps unterscheiden sich darin, dass bei der zweiten Map das FeatureType *osm\_ds:boundary* weggelassen wurde, um die Grenzen von einer externen Quelle laden zu können. SRS und Boundingbox wurden gleich konfiguriert wie bei den jeweiligen FeatureTypes.

---

## GEOWEBCACHE

GeoWebCache wird eingesetzt um die Tiles zwischen zu speichern. Dieser Dienst wird über die Datei */data/geoserver/gwc/geowebcache.xml* konfiguriert. Diese Datei wurde so konfiguriert, dass die beiden Base Maps, welche für den WMS erstellt wurden, gecached werden.

### **Probleme:**

GeoWebCache ignoriert die eingestellte Caching dauer. Auch eine Nachfrage bei der GeoServer-Mailingliste konnte das Problem nicht zufriedenstellend lösen.

## 5.6.2 INSTALLATIONSMANUAL

### PRECONDIDTION

Before the installation can be started, the following applications have to be installed and the preconditions have to be met.

1. Tomcat 5.5 installed.
2. Java 1.6 installed.
3. PostgreSQL 8.3 installed.
4. PostGIS 1.3.5 installed.
5. A database with PostGIS and HStore created.
6. Tomcat has to be running.
7. The Server running OSM-in-a-Box has to be connected to the Internet.

### WINDOWS AND UNIX INSTALATION

The print screens are taken from the Windows installation, but are not very different from the Unix ones. Administrator right are required in order to install OSM –in-a-Box.

1. Unpack *osminabox1.0\_win.zip* for Windows or *osminabox1.0\_unix.zip* for Unix to a folder and navigate to this directory.
2. Start the installer:
  - [a] On **Windows** with the file *installer.bat*
  - [b] On **Unix** with the file *installer.sh*
    - i. On Unix, the execution rights may have to be set on *installer.sh*. The command would be: `chmod +x installer.sh`
3. Set the folder where *osm2gis* should be installed standard is:
  - [a] **Windows**: `C:\Program Files\osm2gis`
  - [b] **Unix**: `/data/osm2gis`

```
Set osm2gis directory [C:\Program Files\osm2gis]:
Archive:  osm2gis_win.zip
inflating: C:/Program Files/osm2gis/osm2gis.jar
inflating: C:/Program Files/osm2gis/config/LoggingConfiguration.xml
inflating: C:/Program Files/osm2gis/config/mapping.xml
inflating: C:/Program Files/osm2gis/config/osm2gis.properties
inflating: C:/Program Files/osm2gis/config/osm_create.sql
inflating: C:/Program Files/osm2gis/config/osmconfig.properties
inflating: C:/Program Files/osm2gis/lib/JSAP-2.1.jar
inflating: C:/Program Files/osm2gis/lib/bzip2.jar
inflating: C:/Program Files/osm2gis/lib/commons-httpclient-3.1.jar
inflating: C:/Program Files/osm2gis/lib/gt-main-2.5.4.jar
inflating: C:/Program Files/osm2gis/lib/gt-metadata-2.5.4.jar
inflating: C:/Program Files/osm2gis/lib/jts-1.9.jar
inflating: C:/Program Files/osm2gis/lib/log4j-1.2.15.jar
inflating: C:/Program Files/osm2gis/lib/postgresql-8.3-604.jdbc4.jar
inflating: C:/Program Files/osm2gis/lib/quartz-1.6.5.jar
inflating: C:/Program Files/osm2gis/manpage.txt
inflating: C:/Program Files/osm2gis/osm2gis.bat
```

Figure 18: Installation osm2gis folder

4. Set the webapps folder of your tomcat installation. This step will take some time, because the installer has to wait until tomcat has deployed 'GeoServer'. Afterwards the installer has to stop tomcat for further modifications. Standard value here is:

[a] **Windows:** C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps.

[b] **Unix:** /var/lib/tomcat5/webapps

```
Set tomcat directory [C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps]:
geoserver\geoserver.war
1 File(s) copied

The Apache Tomcat service was stopped successfully.
```

Figure 19: Installation webapps folder

5. Set the GeoServer data directory where GeoServer stores its configurations and the tiles. Please make sure that there's enough empty disk space available.

**Important:** 2 backslashes have to be used between the folders when installing it on **Windows**.

Standard values are:

[a] **Windows:** C:\\Program Files\\Apache Software Foundation\\Tomcat 5.5\\webapps\\geoserver\\data.

[b] **Unix:** /var/lib/tomcat5/webapps/geoserver/data

```
Set geoserver data directory [C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\data (use \\ between folders):D:\\geoserver C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\WEB-INF\web.xml
1 File(s) copied
```

Figure 20: Installation GeoServer data directory

6. Set the GeoServers admin password. This password is needed to change configurations via web interface.

[a] Standard value is `geoserver`.

```
Set geoserver admin password [geoserver]:test
C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\data\security\users.properties
1 File(s) copied
```

Figure 21: Installation admin password

7. Set the name of the database, which has been created on the preparation steps.

```
Set db name [osm]:test
C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\data\catalog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
```

Figure 22: Installation database name

8. Set the database user.

```
Set db user [osm]:osm
C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\data\ca
talog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
```

Figure 23: Installation database user

9. Set the database password for the database user entered in step 8.

```
Set db password [osm]:test
C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\geoserver\data\ca
talog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
The Apache Tomcat service is starting.
The Apache Tomcat service was started successfully.

You succesfully installed OSM-in-a-Box
Press any key to continue . . .
```

Figure 24: Installation database password

After you completed these steps OSM-in-a-Box is successfully installed on your operating system. You are now ready to use osm2gis to import the OpenStreetMap data into your database. After the initial import you can use GeoServer [5] as you know it.

You can find a demo website with OpenLayers [4] in the subfolder geoserver/www/. It is pre-configured for the use when you import Switzerland into your database.

### 5.6.2.1 USERMANUAL

#### PURPOSE OF THE OSM2GIS UTILITY

The osm2gis utility is a tool, which can be used to import OSM data files into a local database. It also provides the capability to update the local database using the OSM differential update files.

#### PRECONDITIONS

The osm2gis tool is used to import the data from the OSM planet or differential files into the local database. In order to run the application the following conditions have to be achieved:

- PostgreSQL must be running
- A user and database must be created
- The database must be created from a PostGIS template.
- The database must be prepared for using Hstore

How to achieve these requirements is described in the installation manual in chapter 5.6.2

## GENERAL INFORMATION

To use the `osm2gis` tool the command line depends on which platform it should be executed. Replace the `osm2gis` occurrence described in this document with one of the following commands:

- Windows  
`osm2gis.bat`
- Unix/Linux  
`osm2gis.sh`

## CONFIGURATION

For every call of the `osm2gis` utility a database connection has to be established. If the logon values are not specified, the values will be taken from the configuration file `osm2gis.properties`, which is located in the `config` subdirectory. These values can be overwritten with any call of the `osm2gis` utility by specifying the correct parameters:

### Pass the database login information:

```
osm2gis--config -h [hostname] -d [database] -u [user] -p [password]
```

The parameters can be specified at any call, but they will not be saved unless the `config` option is set.

## INITIAL IMPORT

The import can be executed using the following options:

### Import a planet file from a URL:

```
osm2gis--initial-import -t -l [planetfileurl]
```

This will download the planet file from the given URL and import the content into a new database. The `-t` option will force the application to create all necessary tables the application will need.

### Import the planet file from a local path:

```
osm2gis--initial-import -b [planetfile]
```

This will import the planet file from a location on a mapped or local drive.

### Import the planet file data using a bounding box:

```
osm2gis--initial-import -l [planetfileurl] --latMax [latitude] --lonMin [longitude] --latMin [latitude] --lonMax [longitude]
```

The application will only import nodes, which lie within the given bounding box.

## INITIAL IMPORT USING STDIN (WORKAROUND FOR THE BZIP2 PROBLEM)

As mentioned in the software design document of the osm2gis application the used bzip2 java library will not be able to import large planet files. The cause is the planet file generation process by OSM, which uses multiple processor cores to generate the file. The different parts are put together, and that is something the bzip2 implementation cannot handle at this time.

The workaround for this problem is to use the bzcata utility to decompress the planet file and pipe the resulting stream directly into the standard input of the osm2gis application:

```
bzcata [planetfile] | osm2gis --initial-import
```

If no planet file is specified, the utility will automatically listen on the standard input for an OSM xml stream.

**Important:** bzcata is a linux/unix tool. Under windows the cygwin [19] implementation can be used.

## UPDATE SCHEDULING

An update can be scheduled using the following options:

### Schedule an update

```
osm2gis --schedule-update -f [daily/hourly/ minutely] -b [initial differential file name]
```

This will schedule an update with the specified frequency using the given initial differential update file as start file.

### Schedule an update using the last differential update information from the config file

```
osm2gis --schedule-update -f [daily/hourly/ minutely]
```

This will schedule an update with the specified frequency. The last stored next initial differential file name will be used.

### Schedule an update for a specified bounding box:

```
osm2gis --schedule-update -l [planetfileurl] --latMax [latitude] --lonMin [longitude] --latMin [latitude] --lonMax [longitude]
```

This will schedule an update only importing data which lies within the specified bounding box.

## HELP

Help inside the command line can be found by typing.

```
osm2gis --help
```



## 6 ANHANG

### 6.1 ERFAHRUNGSBERICHTE

#### 6.1.1 MICHAEL HUBER

Der Sektor rund um GIS-Systeme war für mich bis dato relativ unbekannt. Auch konnte ich mir nicht vorstellen, dass man solch komplexe und interessante Arbeiten in diesem Bereich durchführen kann. Als ich die Ausschreibung sah setzte ich mich sofort dafür ein, dass wir im Team uns für diese Arbeit bewerben.

Die Arbeit selbst war für mich persönlich sehr interessant. Ich habe mich auch zuvor ein paar Mal gefragt, wie man solche GIS-Daten sinnvoll persistent speichern und korrekt verwalten kann, daher war auch ein Ausführen dieser Arbeit sehr interessant.

Nach ein paar Unstimmigkeiten und Änderungen an der Aufgabenstellung (zuerst sollte das OSM-API nachgebaut werden, dies wurde dann in OSM-in-a-Box umgeändert) konnte es richtig losgehen. Aufgrund des absoluten Neulands mit GIS Systemen mussten wir uns zuerst mühsam und aufwändig in den ganzen Themenbereich OSM und GIS Systemen einarbeiten. Studium der Standards rund um OSG sowie auch die zum Teil interessanten Diskussionen mit der OSM-Community waren sehr aufschlussreich und bereichernd.

Die Verwendung der für mich neuen Technologien wie zum Beispiel PostGIS sowie GeoTools war sehr spannenden und interessant. Ich muss jedoch auch sagen, dass die komplette Einarbeitung in die GIS Welt sehr viel Zeit und Aufwand gekostet hat, denn die Materie nicht immer sehr trivial. Auch die zum Teil schlechten oder gar nicht vorhandenen Dokumentationen rund um OpenStreetMap machten die Sache nicht einfacher. Vieles musste wie Mailingliste oder Forum abgeklärt werden.

Nun ist die Arbeit vollbracht und es ist auch Zeit über die letzten, zum Teil sehr intensiven, 17 Wochen nachzudenken. Die Arbeit im Team war sehr anspruchsvoll. Ich hatte bis dato noch nie in einem dreier Team gearbeitet. Dies war am Anfang nicht immer einfach, denn die Aufgabenteilung wurde durch drei statt zwei Teammitglieder sicherlich nicht erleichtert. Trotzdem oder gerade deswegen habe ich viel Erfahrungen und Eindrücke für die Zukunft sammeln können welche mir in späteren Tätigkeiten sicher eine Hilfe sein werden.

Auf die geleitete Arbeit bin ich recht stolz. Wir konnten alle geforderten Aufgaben sowie ein paar weitere, nicht geforderte Features entwickeln. Für ein weiteres Projekt nehme ich sicherlich meine Eindrücke rund um die Arbeit in einem dreier Team mit. Anders würde ich aus heutiger Sicht höchstens die Dokumentation der Software angehen. Eine frühere Dokumentation wäre sicherlich von Vorteil gewesen, war jedoch in diesem Projekt nicht immer angebracht, weil wir vieles neu entwickeln mussten und vom Voraus den Umfang des Problems kaum abschätzen konnten.

---

## 6.1.2 ROLAND HOF

Im Verlauf des Projektes hatte ich die Möglichkeit mich in diverse neue Technologien einzuarbeiten, vorhandenes Wissen zu vertiefen und praktisch anzuwenden. Dabei habe ich positive wie auch negative Erfahrungen gemacht.

### TEAMGEIST

Die Zusammenarbeit des Teams war zum grossen Teil sehr gut. Aufkommende Probleme wurden schnell erkannt und untereinander kommuniziert. Jeder hat seine Stärken und konnte diese in das Projekt einbringen. Dennoch war es nicht immer einfach. Wir waren uns untereinander nicht immer einig, wie man gewisse Probleme angeht. Da es beim Entwickeln von Software selten einen eindeutigen Lösungsweg gibt, hat es öfters heftige Diskussionen gegeben, wie etwas umgesetzt werden soll. Schlussendlich konnten wir uns aber immer auf eine Lösung einigen.

### ZEITPLANUNG

Das grösste Problem dass wir hatten, war unsere Zeitplanung. Wir wollten die Applikation so konstruieren, dass sie möglichst flexibel, anpassungsfähig und erweiterbar ist. Dies ist uns auch gelungen, dafür mussten wir die Funktionalität etwas reduzieren. Die Zeitplanung hat aber trotzdem gut funktioniert. Wir konnten die eingeplante Woche Reserve aber gut gebrauchen.

### NEU ERLERNTTE TECHNOLOGIEN:

- StAX XML Parsing
- Quartz

### FAZIT

Obwohl mein Gesamtgefühl für das Projekt positiv ausfällt, gibt es dennoch ein paar Dinge die ich das nächste Mal anders machen würde.

- Refactorings früher durchführen
- Design besser durchdenken (auch in Punkten bei denen man meinen könnte dass es klar sei)
- Schnittstellen zwischen den Paketen genauer definieren.
- Einfache architektonische Lösungen implementieren, wenn keine Notwendigkeit für mehr Flexibilität besteht.

---

### 6.1.3 FABIO RENGGLI

#### TEAM

Die Arbeit im Team mit Michael Huber und Roland Hof hat eigentlich sehr gut geklappt. Da wir schon mehrere kleinere Projektarbeiten, mit Michael Huber sogar die Studienarbeit, und sonstige Arbeiten für unser Studium zusammen gemacht haben, wissen wir unterdessen, wie wir miteinander umzugehen haben, damit Kritik an der Arbeit angebracht werden kann, ohne das dies zu einer unprofessionellen Auseinandersetzung führt. Klar waren kleiner Meinungsdivergenzen nicht ganz zu Umgehen. Trotzdem herrschte immer eine professionelle Atmosphäre.

Durch unsere ähnlichen Stundenpläne in diesem Semester arbeiteten wir meistens gemeinsam in einem Raum und konnten uns so bei Fragen gegenseitig sehr gut unterstützen.

#### ZEITPLANUNG

Die Zeitplanung ist uns in diesem Projekt ziemlich gut geglückt. Klar gab es bei einigen Arbeitspaketen kleine Differenzen zwischen den Soll- und Ist-Zeiten. Diese blieben aber fast immer in einem annehmbaren Rahmen.

Ein grosses Problem, welches extrem viel Zeit verschlungen hatte, war dadurch bedingt, dass durch Bugs in der eingesetzten Software grosse Teile der Arbeit, welche ich ins Arbeitspaket ‚SLD‘ gesteckt habe verloren gingen. Ein zweites Problem welches sich in der Zeitplanung bemerkbar macht, ist in dem Arbeitspakete ‚Infrastruktur‘ zu finden. Dieser enorme Zeitunterschied erklärt sich dadurch, dass ich den Server erneut installieren musste, damit ich ein sauberes System hatte.

Durch unsere Aufteilung der einzelnen Arbeitspakete in die Phasen des Projektes und den dazugehörigen Milestones ist es uns auch gelungen die Arbeit einigermaßen gleichmässig auf die gesamten 17 Wochen der Bachelorarbeit zu verteilen.

#### NEU ERLERNTTE TECHNOLOGIEN

- GeoServer
- OpenLayers
- Style Layer Deskriptors

#### FAZIT

Ich denke wir haben die Arbeit im Ganzen sehr gut gemeistert. Trotzdem gibt es Arbeiten die ich anders erledigen würde:

- Von Anfang an eine frische Installation des Servers machen, damit nur Programme installiert sind, welche auch wirklich gebraucht werden.
- Bei grossen Änderungen auf dem virtuellen Server zuerst einen Snapshot erstellen.
- Eingesetzter Software nicht voll vertrauen. Kleine Bugs können leider grosse Auswirkungen haben.

## 6.2 INHALT DER CD

Der Inhalt der CD gliedert sich wie folgt:

Pfad	Dateien
<b>/Dokumente/</b>	
	Gesamtdokumentation.pdf
	Gesamtdokumentation.docx
	Plakat.pptx
<b>/Dokumente/Projektplan/</b>	
	Projektplan.xlsx
	Zeiterfassungen
<b>/Java_doc/</b>	
	Javadoc
<b>OSM-in-a-Box 1.0</b>	
	Installationsdateien für OSM-in-a-Box
<b>Sitzungsprotokolle</b>	
	Alle Sitzungsprotokolle
<b>Source</b>	
	Source Code

## 6.3 ANHANG B GLOSSAR UND ABKÜRZUNGSVERZEICHNIS

Term	Explanation
<b>CentOS</b>	CentOS is a linux distribution based on RedHat
<b>Geom</b>	Database column used to store geospatial information on PostGIS. [16]
<b>Geoserver</b>	The GeoServer project. [5]
<b>GIS</b>	Geographic information system.
<b>GWC</b>	GeoWebCache caching service for tiles.
<b>Mapnik</b>	Tile renderer for OSM. <a href="http://mapnik.org">http://mapnik.org</a>
<b>Node / Point</b>	Terms are equal and define a data type in OSM. See also [6b]
<b>OGS</b>	Open Geospatial is a standards organisation geospatial and location based services. <a href="http://www.opengeospatial.org">http://www.opengeospatial.org</a>
<b>OpenLayers</b>	OpenSource program to display map tiles in a web browser. [4]
<b>OSM</b>	The project OpenStreetMap [6]
<b>Planet-File</b>	Weekly extract of the OSM database in xml format
<b>Relation</b>	OSM primitive data type [6]
<b>SLD</b>	Style Layer Descriptor. Defines how the tiles should look like.
<b>SRS</b>	Spatial referencing system.
<b>stAX</b>	XML Parser [18]
<b>Tiles</b>	Image of a part of the map.
<b>Way</b>	OSM primitive data type [6]
<b>WFS</b>	Web Feature Service: Standardized interface on GeoServer which return raw vector data
<b>WMS</b>	Web Map Service: creates the tiles on GeoServer
<b>Yum</b>	A linux tool to install software from repositories.

## 6.4 ANHANG C LITERATUR- UND QUELLENVERZEICHNIS

### BÜCHER UND ARTIKEL

- [1] Frederik Ramm und Jochen Topf, «OpenStreetMap», 2. Version 2009
- [2] Frederik Ramm und Jochen Topf, «OpenStreetMap», 1. Version Feb. 2008
- [3] Jochen Topf, «OpenStreetMap Data in Standard GIS Formats», whitepaper V 0.2, 04.09.2008

### LINKS UND INFORMATIONEN

- [4] OpenLayers: <http://openlayers.org/>
- [5] GeoServer: <http://www.geoserver.org>
- [6] OpenStreetMap: <http://www.openstreetmap.org>
  - [a] Osmosis, <http://wiki.openstreetmap.org/wiki/Osmosis>
  - [b] Data primitives, [http://wiki.openstreetmap.org/wiki/Data\\_Primitives](http://wiki.openstreetmap.org/wiki/Data_Primitives)
  - [c] Relation multipolygons, <http://wiki.openstreetmap.org/wiki/Relation:multipolygon>
  - [d] Tags <http://wiki.openstreetmap.org/wiki/Tags>
  - [e] Wiki in general <http://wiki.openstreetmap.org>
- [7] Open Geospatial Consortium OpenGIS standards and specification, <http://www.opengeospatial.org/standards>
- [8] JAXB development and documentation, <https://jaxb.dev.java.net/>
- [9] GeoTools / jts documentation and mailing list, <http://geotools.codehaus.org/>
- [10] Quartz documentation, <http://www.opensymphony.com/quartz/wikidocs/Documentation.html>
- [11] Bzip2 documentation, <http://www.kohsuke.org/bzip2/>
- [12] SLD documentation, <http://www.opengeospatial.org/standards/sld>
- [13] Log4J, <http://logging.apache.org/log4j/>
- [14] Junit, <http://www.junit.org/>
- [15] PostgreSQL 8.3 documentation, <http://www.postgresql.org/docs/8.3/interactive/index.html>
- [16] PostGIS 1.5.3 documentation, <http://postgis.refractory.net/documentation/>
- [17] GeoWebCache documentation, <http://geowebcache.org/trac>
- [18] stAX, <http://stax.codehaus.org/Home>
- [19] cygwin, <http://www.cygwin.com/>

## 6.5 ANHANG D EIGENHÄNDIGKEITSERKLÄRUNG

### 6.5.1 MICHAEL HUBER

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.

Ort, Datum:

Name, Unterschrift:

### 6.5.2 ROLAND HOF

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.

Ort, Datum:

Name, Unterschrift:

### 6.5.3 FABIO RENGGLI

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.

Ort, Datum:

Name, Unterschrift: