OPENSTREETMAP-IN-A-BOX 1.0

GESAMTDOKUMENTATION

Gruppenmitglieder: *Meier Andreas*

Zimmermann Joram

ABSTRACT

INTRODUCTION

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 Current base map products are either rather expensive and sometimes outdated or not available for certain regions (i.e. cadastral data or Google Maps).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. OSM itself is run by many open source tools with a bunch 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.

APPROACH

In prior students work the OpenStreetMap-in-a–Box software was created. An easy to use setup installs OSM "out of the box" as a dedicated map server offering well-known OGC web services. OSM-in-a-Box includes several parts as follows:

- A fully configurable (Schema Mapping File) osm2gis import converter (Java) which imports OSM data and inserts the relevant part of it in the database PostgreSQL/PostGIS/geospatial database schema).
- A spatial information server with geographic web services (GeoServer), like WMS, Tiling/Caching and WFS (read- only).
- Due to the configuration complexity of the osm2gis and GeoServer software, a consistency check can be run with the osm2gis software. This generates a statistic report of the whole configuration.
- A website (showcase) to demonstrate the project.

RESULT

The following changes in this bachelor thesis led to the OSM-in-a-Box 1.0 release:

- osm2gis import converter:
  - More OSM data can be imported by the support of entity to entity relations.
  - Keeping the database up to date by downloading and importing the regularly released differential update files.
- GeoServer:
  - Update of GeoServer to current version 2.0. Revise of theconfiguration (map presentation, caching, services).
- osm2gis consistency check:
  - Update of the consistency check to support the new GeoServer version. Extend the check to show selfinconsistency of the osm2gis Schema Mapping File.
- Showcase:
  - Update of the website to support the above changes.

Web: http://dev.ifs.hsr.ch/osminabox

MANAGEMENT SUMMARY

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. OSM itself is run by many open source tools with a bunch 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.

In prior students work the OpenStreetMap-in-a–Box software was created. An easy to use setup installs OSM "out of the box" as a dedicated map server offering well-known OGC web services. OSM-in-a-Box includes several parts as follows:

- A fully configurable (Schema Mapping File) osm2gis import converter (Java) which imports OSM data and inserts the relevant part of it in the database PostgreSQL/PostGIS/geospatial database schema).
- A spatial information server with geographic web services (GeoServer), like WMS, Tiling/Caching and WFS (read- only).
- Due to the configuration complexity of the osm2gis and GeoServer software, a consistency check can be run with the osm2gis software. This generates a statistic report of the whole configuration.
- A website (showcase) to demonstrate the project.

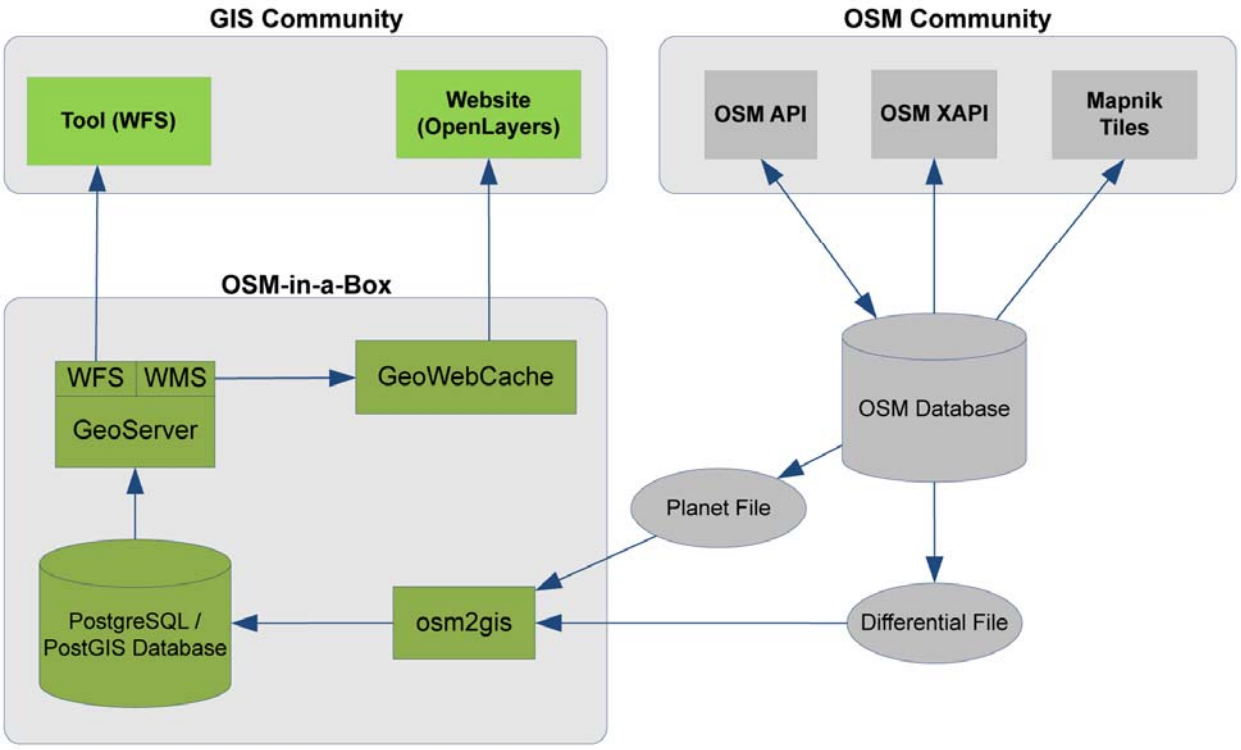


Figure 1: OSM-in-a-Box as link between OSM and GIS

GOALS

The OpenStreetMap-in-a-Box to software needs to be upgraded on different parts. A major goal of this project was the support of entity to entity relations, differential update and update of the 3rd party software to the newest versions. Changes are:

- osm2gis initial import:
  - More OSM data can be imported by the support of entity to entity relations.
- osm2gis differential update:
  - The database is kept up to date by downloading and importing the regularly released differential update files on a daily, hourly or minutely basis.
- osm2gis consistency check:
  - Update of the consistency check to support the new GeoServer version.
  - Extend the check to show self-inconsistency of the osm2gis Schema Mapping File.
- osm2gis Schema Mapping File:
  - Extension of the Mapping File to import and update more data (meadows, street-, train routes, etc.).
  - Import of data to be used by IndoorGuide4Android.
- GeoServer:
  - Update of GeoServer to current version 2.0.
  - Revise of the configuration (map presentation, caching, services, etc.).
- Showcase:
  - Update of the website to support the above changes.
  - Update of the MapCompare website with additional map sources.
- Mobile WMS Viewer:
  - Implementation of a Mobile WMS Viewer for android mobiles.
- Version 1.0
  - Release of a stable version 1.0 of the software

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
osmosis	Most powerful tool for OSM. Mostly used to import / export data in OSM format	http://wiki.openstreetmap.org/wiki/Osmosis
osm2pgsql	osm2pgsql is a utility program that converts OpenStreetMap (OSM) data into a PostgreSQL format.	http://wiki.openstreetmap.org/wiki/Osm2pgsql

APPROACH

Most required knowledge was already gained due to the prior work (semester thesis) on this project. The different goals were divided to the participants according to their prior work on the subject. The goals were prioritized for the major ones to be certainly attained. The support for entity to entity relations had the biggest impact on the software and was done first. Rewriting and refactoring most of the parts in the import process were needed as well as an extension of the Schema Mapping File. The upgrade to GeoServer 2.0 brought structural changes for the consistency check and was also implemented at the beginning of the project. In the second half of the remaining time the differential update support was implemented as well as many smaller and bigger bug fixes within the GeoServer configuration. Nearly at the end of the project a new GeoServer version 2.0.2 was released with was included in the OSM-in-a-Box software immediately. In fact it brought some helpful changes and bug fixes along. The software was tested continuously after changes were made and blackbox tests at the end guaranteed a stable version 1.0 release.

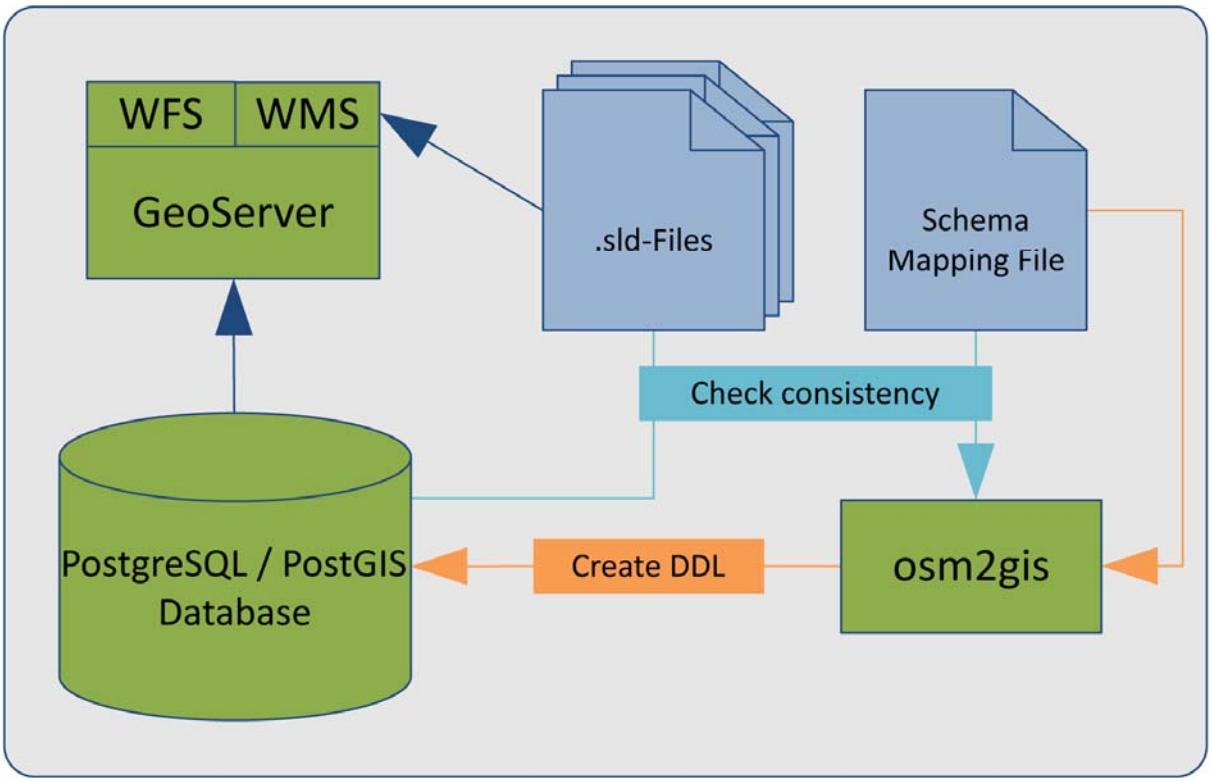


Figure 2: Preparation phase

INVOLVEMENT

The OpenStreetMap-in-a-Box team consists of Andreas Meier and Joram Zimmermann.
The progress of the project was controlled and emerging question was discussed on a weekly meeting with supervising Prof. Stefan Keller.

OpenStreetMap specific question were discussed with Frederik Ramm from www.geofarbrik.de and the OSM Mailing List.

RESULTS

The results in this bachelor thesis are the following:

- an import converter which supports all the OSM datatypes
- the import of planet and differential update files
- support of a fully configurable destination scheme (db schema) with relations between tables using a join table
- the possibility to check the configuration of all involved programs (Schema Mapping File, database and GeoServer styles)

The whole install package of OSM-in-a-Box consists of following components:

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

GOAL ATTAINMENT

All primary goals have been achieved. Some subordinate tasks have been added and fulfilled like the support of POIs needed by the IndoorGuide4Android team.

The implementation of a mobile WMS viewer for android mobiles was shortened to an evaluation document for available libraries. The time profits from this decision helped in improving the GeoServer configuration even more.

EXTERNAL RESSOURCES

For this project several external libraries were used 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.

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:

- Design work for more data to be shown on the website (shops, ferries, etc.), search functionality of relations (like S-Bahn S8), street names.
- Intelligent conversion of tag values (population=”about 500” is discarded right now if the population column excepts an integer).
- Download of missing data during an initial import if wanted.
- Create an import summery after an import job is complete.
- Members of relations which have no own mapping should be imported as well if the relation is mapped to a database table and inserted in a own table.
- If multiple mappings with the same table name apply to an OSM entity, only the last assigned mapping is used for inserting in the database. Multiple entries might be wanted.
- Boost performance by looking up primary keys from relations and their members when inserting them instead of separate selects.
- Include the Nodes lat / lon values to the way_temp table so they don’t need to be looked up a second time when a way is used.
- DiffUpdate files can contain the same OSM entity multiple times which can produce errors when processing them.
- Relations can’t be deleted in Potlach but all their Tags and Members can be removed. Such relations should be deleted instead of modified in the database.

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

0 DOCUMENTINFORMATION

0.1 CHANGE HISTORY

Datum	*Version*	*Änderung*	*Autor*
16.06.2010	0.1	Dokument übernommen	ameier
16.06.2010	0.2	Dokument überarbeitet	ameier
17.06.2010	0.3	Einfügen aller Kapitel	ameier/ jzimmerm
18.06.2010	0.4	Review	ameier/ jzimmerm

0 INHALT

1 AUFGABENSTELLUNG

Autoren/Studenten:	Andreas Meier, Joram Zimmermann
Verantwortlicher/ Betreuer:	Prof. Stefan Keller, HSR, Abt. Informatik
Partner (Firma oder Verwaltung), externer Betreuer:	(OSM und GIS Open Source Community

1.1.1 EINFÜHRUNG

OpenStreetMap-in-a-Box dient der Bereitstellung von Hintergrund-Karten für interaktive Kartenapplikationen im Web und auf Mobiles. Im Gegensatz zu bekannten Kartendiensten wie beispielsweise Google Maps bietet diese Lösung zwei Vorteile: Erstens die Kontrolle über die Verfügbarkeit des Services (da 'in-house') und zweitens die Möglichkeit, eine an individuelle Bedürfnisse angepasste Kartengrafik zu konfigurieren.

Die Daten stammen vom OpenStreetMap-Projekt (kurz OSM), dem "Wikipedia" der Landkarten. Diese Karten, bzw. Geodaten sind oft detaillierter und aktueller als vergleichbare Produkte. Dies ist möglich durch die GPL-artige Lizenz (CC-by-SA) der Daten.

In der vorangegangenen Semesterarbeit mit dem gleichen Namen wurde eine Server-Applikation mit Datenbank, mit Geo-Webservices und mit zwei eigenen Websites (Webkarte und Webkarten-Vergleich mit Google & Co.) erstellt. Die Java-Software steht unter einer modified BSD License.

1.1.2 AUFGABENSTELLUNG

Der im Vorgängerprojekt hauptsächlich entwickelte Konverter, der die bestehenden OSM-Daten in eine frei definierbare relationale Datenstruktur importiert, soll konsolidiert und nochmals erweitert werden. So soll z.B. die Konfiguration auch mit Relationen umgehen können. Dabei sollen auch die Konsistenztests angepasst werden, welche die Übereinstimmung von Zielschema und Schema-Mapping sowie von Zielschema und GeoServer-Konfiguration prüfen (das Quellschema ist von OSM vorgegeben).

Weitere Erweiterungen betreffend den inkrementellen Import und die definitiven Konfiguration des wichtigen Tile Map Services (GeoWebCache). Vorab muss der Code den aktuellsten Versionen vom GeoServer angepasst werden.

Im letzten Drittel soll die Serverkomponente durch einen Mobilen WMS Client (WMS-Viewer, MobilePOIGuide) auf Basis eines Android-Handys ergänzt werden.

Lieferdokumente (englisch wo angegeben, sonst deutsch):

- Installationsanleitung (englisch).
- Bedienungsanleitung, falls nötig.
- Gesamter compilier-bereiter Sourcecode (englisch) inkl. Programmdokumentation sowie als Download genkennzeichnetes Zip-File mit ausführbarem Bytecode.
- Technischer Bericht und SW-Engineering-Dokumentation.
- Allfällige Dokumente gemäss Vorgaben der Abt. I. (z.B. Kurzfassung, Broschüre, Poster).

1.1.3 HINWEISE

- Die Arbeitsweise ist agil, mit Unit-Tests und kontinuierlichem Build. Es wird Wert auf ausgetestete Software gelegt.
- Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet, d.h. es wird eine Arbeitsleistung von mind. 360 Stunden pro Person erwartet.

1.1.4 RANDBEDINGUNGEN, INFRASTRUKTUR, TERMINE UND BEURTEILUNG

- Randbedingungen Hardware/OS:
  - Server HW: keine Vorgaben
  - Server OS: Linux
  - Client HW und OS: Android
- Software
  - Server: Java SE und EE, GeoServer, PostgreSQL/PostGIS
  - Client: Android Java
  - Eclipse IDE, ANT, Hudson, Trac
- Termine und Beurteilung: gemäss Angaben auf www.hsr.ch. Beurteilung mit besonderem Augenmerk auf die Implementation.

Rapperswil, 11. März 2010

2 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 verarbeitet (Kacheln oder Tiles genannt). Die generierten Kacheln können über einen Service mittels Koordinaten und Zoomlevel abgefragt werden.

2.1 PROBLEMSTELLUNG

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.

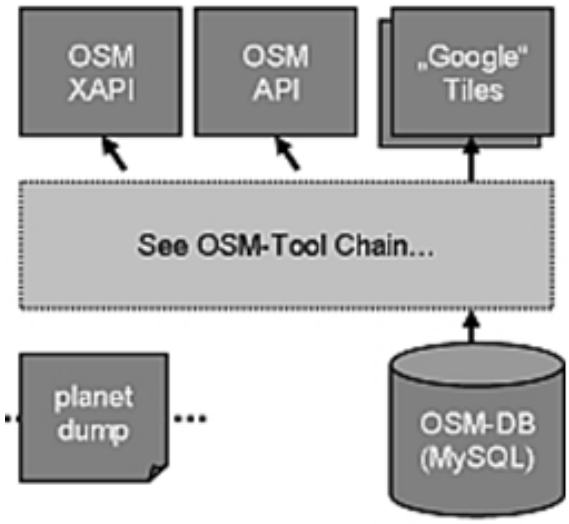


Abbildung 1: OSM-Tool-Chain

2.2 ZIEL DER ARBEIT

Das Ziel des Projektes ist es, 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 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).

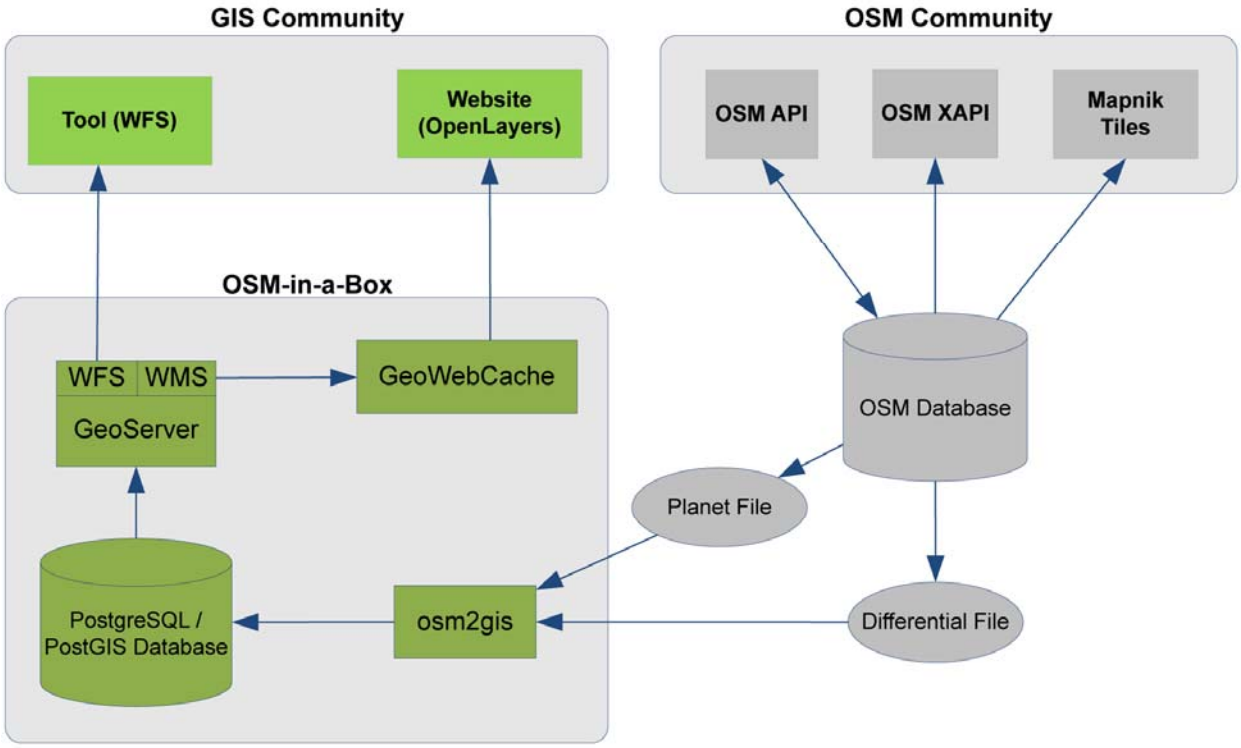


Abbildung 2: Ausgangslage (rechts) mit neuen Erweiterungen (links)

2.3 DIE WICHTIGSTEN ANFORDERUNGEN

Funktionalität	Beschreibung/Bemerkungen	Priorität	Abhängigkeit
OSM-Daten Import	Die Daten der öffentlichen OSM-Datenbank sollen auf möglichst effiziente Weise in die lokale Datenbank importiert werden können.	Hoch	
OSM-Daten Aktualisierung	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
OSM-Daten Konvertierung	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
GeoServer WFS read	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
Website (WMS Client)	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
GeoWebCache	GeoWebCache soll standardmässig nach Installation verfügbar sein.	Hoch	GeoServer WMS read

2.4 EVALUATION MOBILE WMS VIEWER

Die Implementation eines WMS Viewers für Android Mobiles wurde während der Bachelorarbeit zugunsten weiterer Arbeiten am GeoServer in eine Evaluation vorhandener Libraries umgewandelt. Das Ergebnis dieser Evaluation ist in diesem Kapitel beschrieben.

2.4.1 VERFÜGBARE OPENSOURCE WMS-VIEWERS FÜR ANDROID

- Mobiledroidgis - http://code.google.com/p/mobiledroidgis/
- MGMaps Lib - http://developers.cloudmade.com/projects/show/j2me-lib-android
- gvSIG mini - https://confluence.prodevelop.es/display/GVMN/Home
- AndNav2 - http://code.google.com/p/osmdroid/

Liste von http://www.gis.hsr.ch/wiki/WMS#WMS-Clients_f.C3.BCr_Web-Applikationen

- Pois R Us - http://dev.ifs.hsr.ch/osminabox/browser tags Mobile_V0.1

2.4.2 KRITERIENKATALOG UND GEWICHTUNG

Kriterium	Gewicht (1-5)
View WMS Layer **Anzeigen von WMS Layern**	5
POI Search **Suchen nach POIs**	4
Add WMS Layer **Hinzufügen von anderen WMS Layern**	5
Kompass **Intergrierte Kompassfunktionalität**	2

2.4.3 MOBILEDROIDGIS

Details	Kriterium	Gew.	Punkte	Total
Googlemaps werden standardmässig gebraucht	View WMS Layer	5	4	20
Suche nach POIs ist hier nicht möglich, da der Schwerpunkt auf die Verarbeitung von Sensordaten liegt.	POI Search	4	0	0
Keine Möglichkeit um eigene WMS Layers hinzuzufügen	Add WMS Layer	5	0	0
Nicht vorhanden	Kompass	2	0	0

2.4.4 MGMAPS LIB

Dies ist nur eine Library also nicht für die Evaluation geeignet.

2.4.5 GVSIG MINI

Details	Kriterium	Gew.	Punkte	Total
Mehrere WMS-Layers stehen standardmässig schon zur Verfügung.	View WMS Layer	5	10	50
Es kann nach POIs gesucht werden, die eigene Position wird auch angezeigt.	POI Search	4	8	32
Es können beliebige WMS Layer hinzugefügt werden.	Add WMS Layer	5	10	50

Nicht vorhanden.	Kompass	2	0	0

2.4.6 ANDNAV2

Details	Kriterium	Gew.	Punkte	Total
Zeigt die Openstreetmap-Daten an.	View WMS Layer	5	6	30
Suche nach POI ist möglich, auch die eigene Position wird angezeigt.	POI Search	4	8	32
Hinzufügen eines eigenen WMS-Layer ist nicht möglich.	Add WMS Layer	5	0	0
Nicht vorhanden.	Kompass	2	0	0

2.4.7 POIS R US

Dies war eine Studienarbeit im Herbstsemester 2009.

Probleme beim Programmstart, kommt nur die Fehlermeldung “Network error”. Wird eine Statische WMS URL im Code sein.

2.4.8 SCHLUSSFOLGERUNG

	MobileDroidGIS	MGMAPS Lib	gvSIG mini	AndNav2	Pois R us
Punkte	20	0	132	62	0
Entscheidung			Sieger		

Die Evaluation zeigt klar, dass gvSIG mini die Android Applikaiton ist, mit der man am besten eigene WMS Layers anzeigen kann.

2.4.9 MOBILE TEIL WIRD ABGEBROCHEN

Aufgrund der Evaluation der vorhandenen Android WMS Viewer wurde beschlossen, dass dieser Teil der Arbeit zu viel Zeit in Anspruch nehmen würde um schlussendlich ein angemessenes Resultat zu erhalten.

Daher wurde an der wöchentlichen Sitzung vom 28.04.2010 beschlossen, diesen Teil der Bachelorarbeit abzubrechen und die vorhandene Zeit in den GeoServer und der Installationsroutine des OpenStreetMap-in-a-Box Projektes zu investieren. Dies hat zum Ziel, dass die neuen Anforderungen bezüglich der importierten Relationen mit in den Showcase aufgenommen werden können.

2.5 ERGEBNISSE

2.5.1 EINBETTUNG DES OSM-IN-A-BOX SERVERS

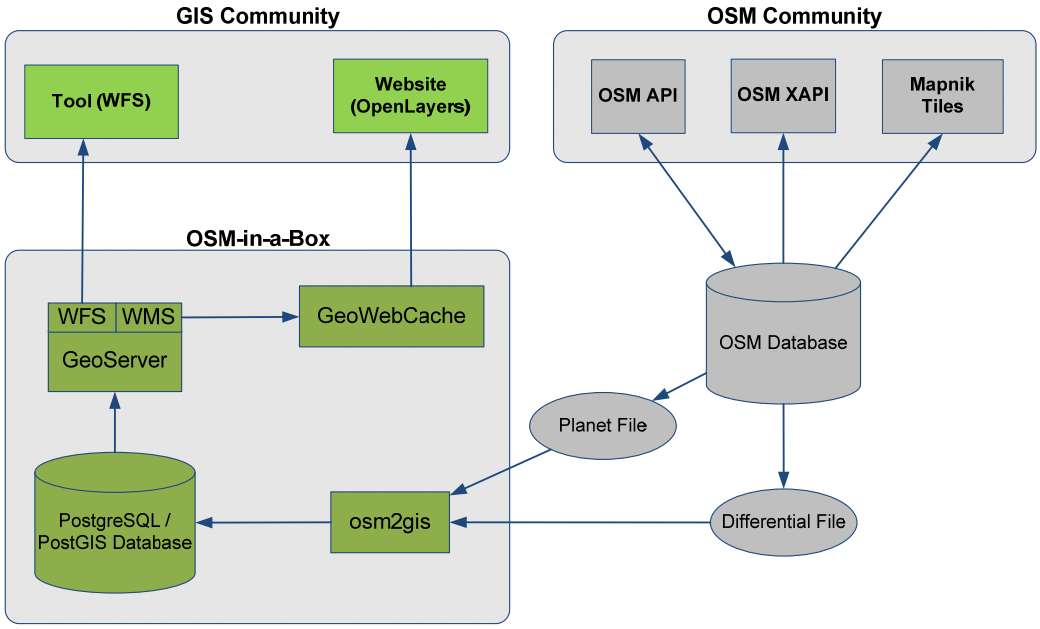


Abbildung 3: Ausgangslage (rechts) mit neuen Erweiterungen (links)

Komponente	Beschreibung/Bemerkungen
OSM Database	Die von in OSM erfassten Daten werden in dieser Datenbank abgelegt.
OSM API	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.
OSM XAPI	Das XAPI ist eine Ergänzung zum OSM-API, welches verbesserte Abfragen und Suchoption zur Verfügung stellt. Das XAPI ist read-only.
Mapnik Tiles	Mapnik ist ein externes Programm welches aus den OSM-Daten Kartenausschnitte (Kacheln oder Tiles) generiert.
Planet File	Das Planet-File beinhaltet sämtliche OSM-Daten im XML Format beschrieben. Planet-Files werden von OSM wöchentlich generiert.
Differential File	Täglich, stündlich oder minütlich generierte XML Dateien von OSM welche die Änderungen innerhalb dieser Zeitspanne wiedergeben.
PostgreSQL / PostGIS Database	Die PostgreSQL-Datenbank ist der zentrale Teil des OSM-in-a-Box Servers. Die OSM-Daten werden darin in einer GIS konformen Struktur abgelegt.
GeoServer	Der GeoServer ist ein Webserver der unter Tomcat läuft. Er bereitet die Daten aus der PostgreSQL-Datenbank auf und generiert die Kartenausschnitte.
osm2gis	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.
WFS	Web Feature-Service ist ein OGC-Standard für die Abfrage von Vektordaten via http.
WMS	Web Map-Service ist ein OGC-Standard für die Abfrage von Kartenausschnitten über http.
GeoWebCache	Cached die Kartenausschnitte, die vom GeoServer generiert werden und reduziert dadurch die Antwortzeit bei der Abfrage von Tiles (Kartenausschnitte).
Tool (WFS)	Dies kann irgendein Tool sein, welches via WFS auf die von GeoServer zur Verfügung gestellten Daten zugreift.
Website (OpenLayers)	OpenLayers ist eine JavaScript Bibliothek welche es ermöglicht, auf einfache Weise eine dynamische Karte in eine Website zu integrieren.

2.5.2 VOLLE KONFIGURATIONSMÖGLICHKEITEN

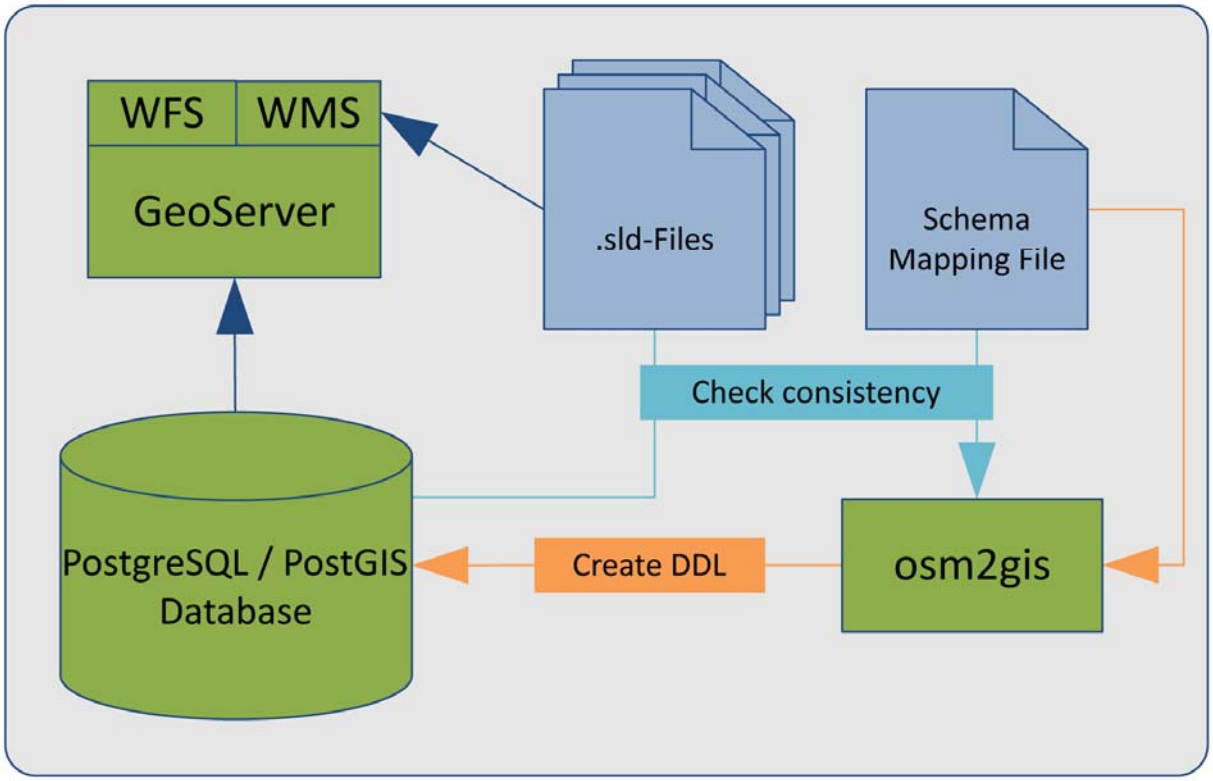


Abbildung 4: Vorbereitungsphase, Konfiguration

Komponente	Beschreibung/Bemerkungen
Schema Mapping File	File um das Zielschema (PostgreSQL/PostGIS DB) sowie die Mappings von den OSM-Daten zu dem Zielschema zu konfigurieren.
osm2gis	Ist eine Java Applikation welche neben dem OSM-Daten Import die Konsistenz zwischen Schema Mapping File, .sld-Files und Datenbank prüfen kann. Das Ergebnis wird anhand eines Reports ausgegeben.
.sld-Files	Styled Layered Descriptor files mit dem die Darstellung der GeoDaten im GeoServer konfiguriert werden kann.
PostgreSQL / PostGIS Database	Die PostgreSQL-Datenbank ist der zentrale Teil des OSM-in-a-Box Servers. Die OSM-Daten werden darin in einer GIS konformen Struktur abgelegt.
GeoServer	Der GeoServer ist ein Webserver der unter Tomcat läuft. Er bereitet die Daten aus der PostgreSQL-Datenbank auf und generiert die Kartenausschnitte.
WFS	Web Feature-Service ist ein OGC-Standard für die Abfrage von Vektordaten via http.
WMS	Web Map-Service ist ein OGC-Standard für die Abfrage von Kartenausschnitten über http.

2.5.3 UNTERSTÜTZUNG SÄMTLICHER OSM DATENTYPEN

Es werden alle Datentypen von OpenStreetMap unterstützt.

Datentypen:

- Node (http://wiki.openstreetmap.org/wiki/Node)
- Way (http://wiki.openstreetmap.org/wiki/Way)
- Relation (http://wiki.openstreetmap.org/wiki/Relation)

2.5.4 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. Sie 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 -u http://planet.openstreetmap.org/planet-090415.osm.bz -m config/mappingconfig.xml

Das Planet File wird vom OSM Server heruntergeladen und die Applikation importiert den Inhalt in die lokale PostgreSQL Datenbank, nach den Mappingregeln die in dem Schema Mapping File definiert wurden.

2.5.5 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, welcher 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 Files bis hin zum aktuellsten importiert (sogenannte “catch-up“ Funktion).

2.5.6 OSM2GIS ARCHITEKTUR

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

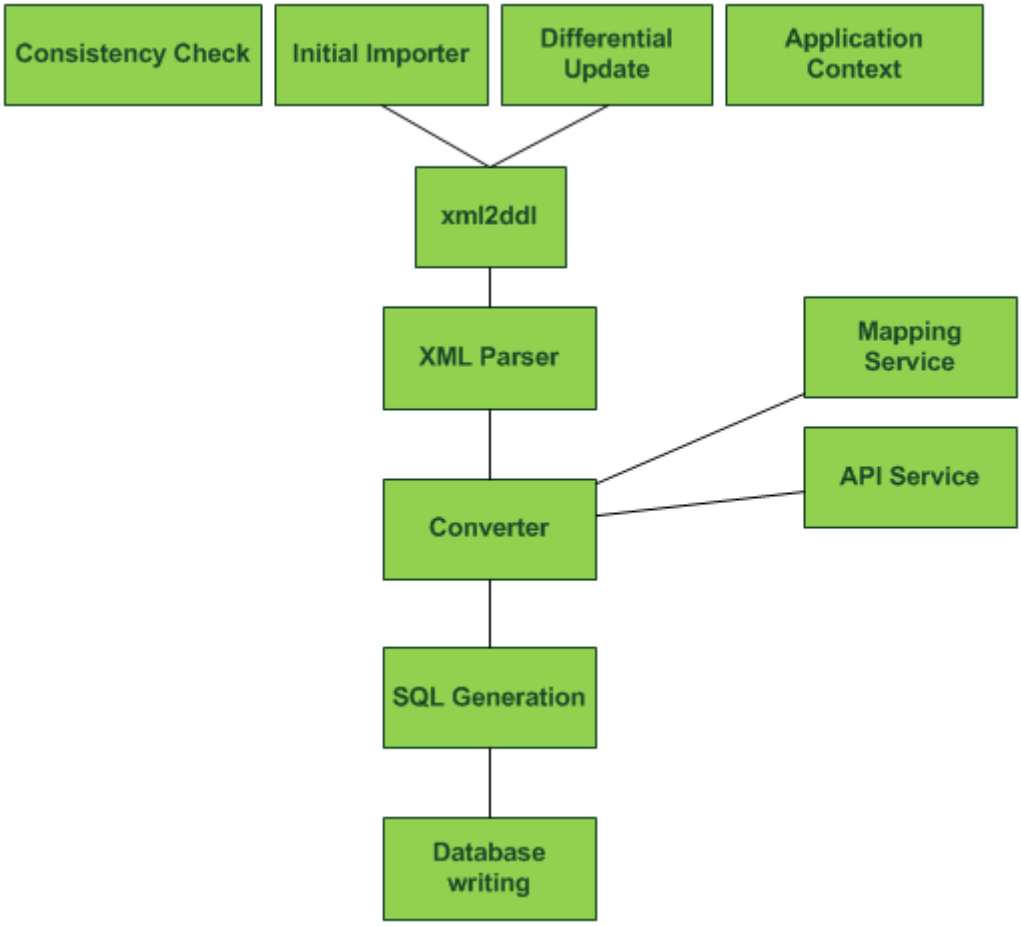


Abbildung 5: osm2gis Software Komponenten

Komponente	Beschreibung/Bemerkungen
Initial Importer	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.
Consistency Check	Die Konsistenzprüfung der die Konsistenz zwischen Schema Mapping File, GeoServer und Datenbank prüft.
Xml2ddl	Xml2ddl generiert falls nötig ein neues DDL um Datenbankänderungen vorzunehmen.
ApplicationContext	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.
XML Parser	Der XML Parser liest die XML Daten eines OSM Planet oder Differential Files.
Converter	Im Converter Teil der Applikation werden die gelesenen XML Daten für die lokale Datenstruktur aufbereitet.
Mapping Service	Wird vom Converter verwendet um die gelesenen Daten einer Tabelle zuzuweisen.
API Service	Der API Service wird vom Converter dazu verwendet unvollständige Daten über das OSM API abzufragen und zu vervollständigen.
SQL Generation	In diesem Teil der Applikation werden die SQL Statements generiert.
Database Writing	Die Daten werden in die PostgreSQL Datenbank geschrieben.

2.5.7 GEOSERVER

Um die importierten Daten auch nutzen zu können, wird GeoServer eingesetzt. GeoServer ist ein OpenSource Server für die Bereitstellung von geospatial Daten. Der Server kann beliebige Daten von jeder spatial data source anhand von Standards veröffentlichen.

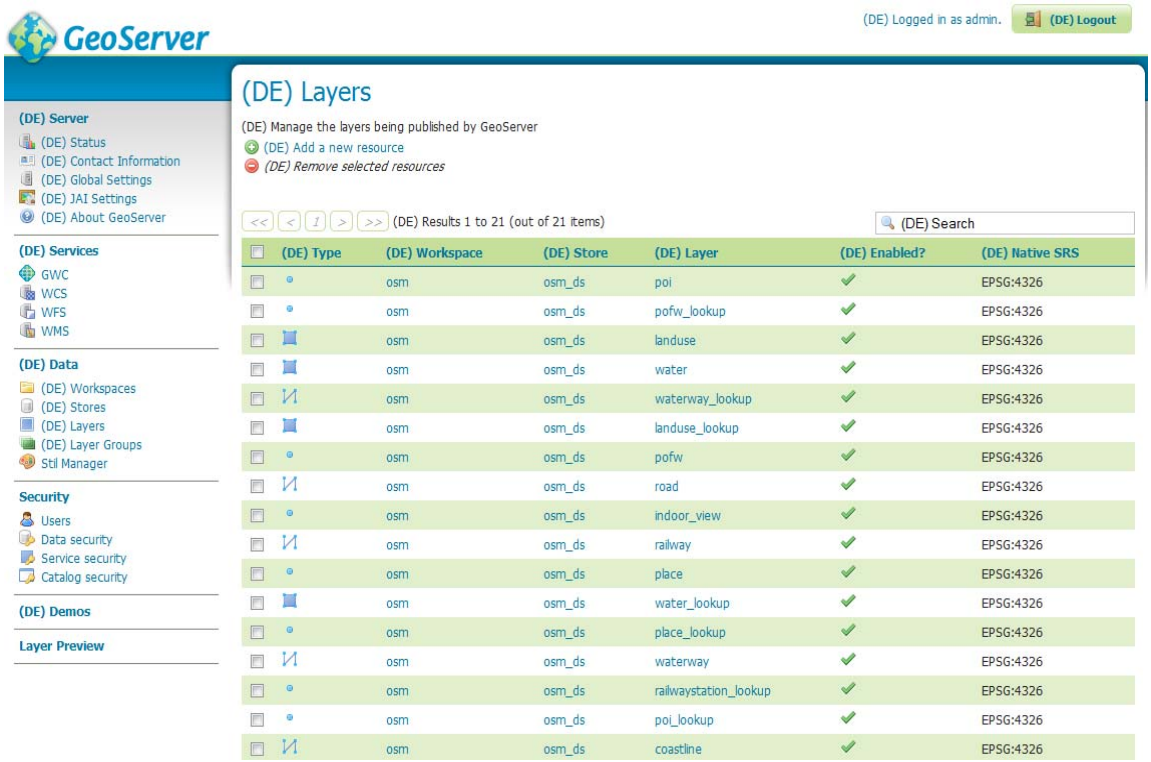


Figure 3: GeoServer 2.0.2

GeoServer 2.0.2 ist im Installationspaket von OSM-in-a-Box 1.0 enthalten und alle Funktionen wurden auf diese GeoServer Version angepasst. Zudem kommt der Server mit einem vollständig vorkonfigurierten Workspace mit dazugehörigen Styles, die auf das mitgelieferte Schema Mapping File zugeschnitten sind.

GeoServer 2.0.2 stellt folgende Services zu Verfügung:

- WCS
  - 1.0.0
  - 1.1.1
- WFS
  - 1.0.0
  - 1.1.0
- WMS
  - 1.1.1

Weitere Informationen: http://www.geoserver.org/

2.5.7.1 GEOWEBCACHE

In der Installation von OSM-in-a-Box 1.0 ist der GeoWebCache standardmässig aktiviert. Der GeoWebCache dient dem Caching der generierten Tiles, dies ermöglicht ein schnelles Laden des Kartenmaterials auf der Website.

2.5.8 WEBSEITE

Um die importierten Daten auch darstellen zu können, wurde in einer Diplomarbeit im Herbstsemester 2009 ein Showcase erstellt. Dieser wurde erweitert, sodass er mit dem GeoWebCache funktionieren kann.

Mit einer Mapcompare Seite können zwei unterschiedliche Datenquellen für Kartenmaterial miteinander verglichen werden.

2.5.8.1 SHOWCASE

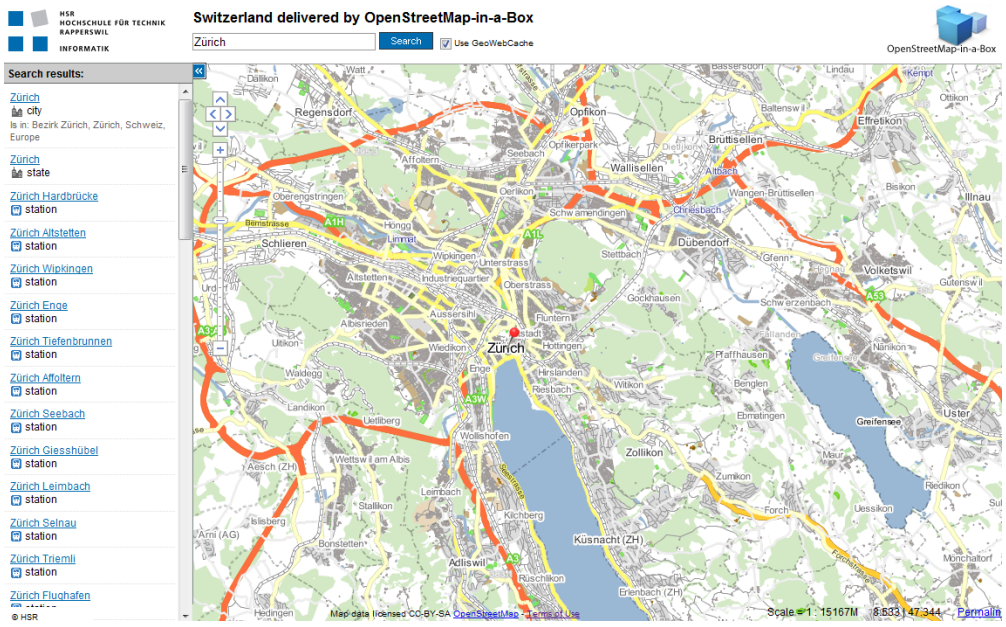


Figure 4: Showcase

2.5.8.2 MAPCOMPARE

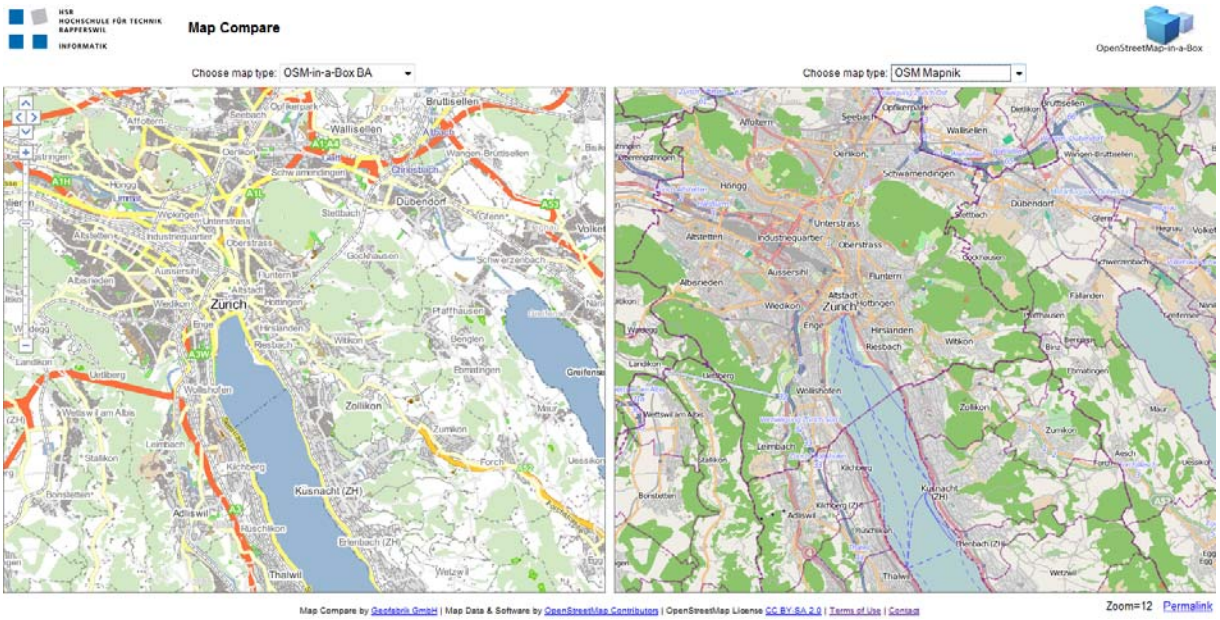


Figure 5: Mapcompare

2.6 SCHLUSSFOLGERUNGEN

Das Resultat ist ein Installer der für Windows oder Unix heruntergeladen werden kann. Mit diesem Installer kann man OSM-in-a-Box "out-of-the-Box" installieren. Das Paket enthält den GeoServer 2.0.2 und den osm2gis import Konverter sowie eine Webseite bestehend aus einem Showcase und Mapcompare. Mit dem osm2gis import Konverter kann man jeden beliebigen OSM Datentyp auf ein beliebiges Zielschema mappen. Nachdem ein Initial-Import gemacht wurde, möchte man die Daten "up-to-date" halten. Dafür gibt es den scheduled update, der auch anhand des osm2gis import Konverter gestartet werden kann.

Das Projekt konnte erfolgreich abgeschlossen werden. Alle primären Ziele wurden erreicht. Der Mobile WMS Viewer Teil wurde nach der Evaluation abgebrochen, sodass noch mehr Zeit für Tests und GeoServer Konfiguration investiert werden konnte.

2.7 WEITERENTWICKLUNG

Wie im Kapitel 2.5 Ergebnisse ersichtlich, haben wir die primären Ziele des Projekts erreicht. Nachfolgende Arbeiten könnten dieses Projekt weiter verbessern. Unserer Meinung nach können folgende Punkte in zukünftigen Arbeiten gemacht werden:

- Designarbeiten bezüglich Darstellung von mehr Daten im Showcase (Shops, Fähren, etc.), Suchfunktionalität von Relations (wie zum Beispiel S-Bahn S8) und Strassennamen.
- Intelligente Übersetzung von tag-values (population="about 500" wird ignoriert falls die Population Kolonne nur integer Datentypen akzeptiert).
- Downloaden von unbekannten Daten während dem Initial-Import, vorausgesetzt dass man dies auch möchte.
- Eine Importzusammenfassung nachdem der Import fertig ist.
- Members von Relations die kein Mapping haben, sollten in eine vordefinierte Tabelle gemapped werden.
- Wenn mehrere Mappings mit demselben Tabellenname zu einer OSM Entity gehören, wird nur das letzte Mapping genommen um die Daten in die Tabelle zu schreiben. Unterstützung mehrerer Mappings wäre sinnvoll.
- Lat / lon Werte von Nodes in die way_temp Tabelle speichern, sodass man nicht ein zweites Mal nach diesen Werten suchen muss.
- DiffUpdate Files können dieselben OSM Entity mehrmals beinhalten, dies kann zu Fehlern führen.
- Tags und Members von Relations können über Potlach gelöscht werden, die Relation selbst jedoch nicht. Diese Relations sollten statt editiert, aus der Datenbank gelöscht werden.

Mehr Information auf: http://dev.ifs.hsr.ch/osminabox

2.8 DANKSAGUNG

Unser Dank für die Unterstützung an dieser Bachelorarbeit geht an:

- Prof. Stefan Keller für die Hilfestellung bei den wöchentlichen Meetings.
- Adrian Geiter, Christoph Egger für das Bereitstellen eines realistischen Use Cases.
- OSM Community
- An alle Studenten im Bachelorarbeitszimmer für die produktive Arbeitsatmosphäre.

3 TEIL II SW-PROJEKTDOKUMENTATION

3.1 ANFORDERUNGSSPEZIFIKATION

3.1.1 EINFÜHRUNG

3.1.1.1 ZWECK

Die Anforderungen des Kunden resp. Der Aufgabenstellung werden in der Requirements Spezifikation festgehalten. Die erkannten Anforderungen werden hier detailiert aufgezeigt und analysiert. Die funktionalen Anforderungen werden soweit möglich mithilfe von UseCases dargestellt.

3.1.1.2 ÜBERSICHT

Im ersten Teil des Dokuments werden die funktionalen und nicht funktionalen Anforderungen an das Projekt festgehalten. Im zweiten Teil werden diese Anforderungen mit Hilfe von UseCases detailierter erfasst.

3.1.2 ALLGEMEINE BESCHREIBUNG DER ANFORDERUNGEN

3.1.2.1 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, mitte) und in einem Mobile Phone (Handy, rechts)

3.1.2.2 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.

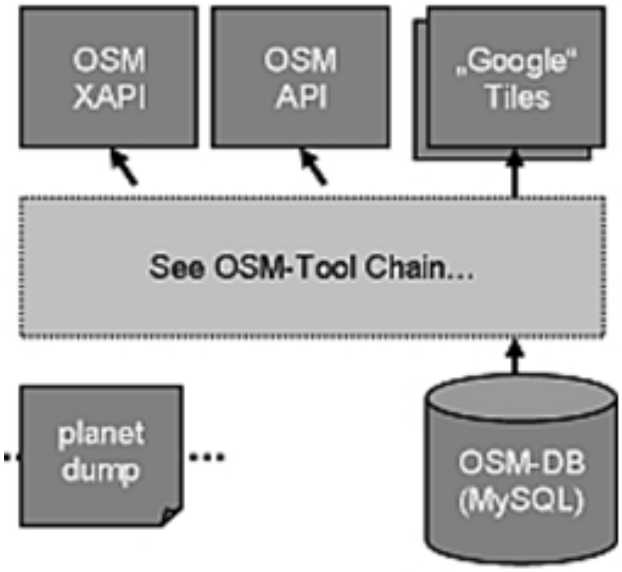


Abbildung 8: Ausgangslage

Im Vorgängerprojekt wurde ein Konverter (osm2gis) geschaffen, welcher die Daten von OSM importiert und in eine für die GIS-Welt standardisierte Datenstruktur konvertiert. Mittels Updatefunktion können automatisch die von OSM bereitgestellten Differentialfiles heruntergeladen und in die bestehende Datenstruktur übernommen werden. Neben dem osm2gis-Konverter wurde ein GeoServer vorkonfiguriert, der diverse Interfaces und APIs, namentlich Tiles, WMS (Web Map Service) und WFS (Web Feature Service) zu Verfügung stellt. Das Ziel war eine einfache out-of-the-box Installation zu ermöglichen, um Kartenmaterial auf einer Website darzustellen oder via Services abrufen zu können.

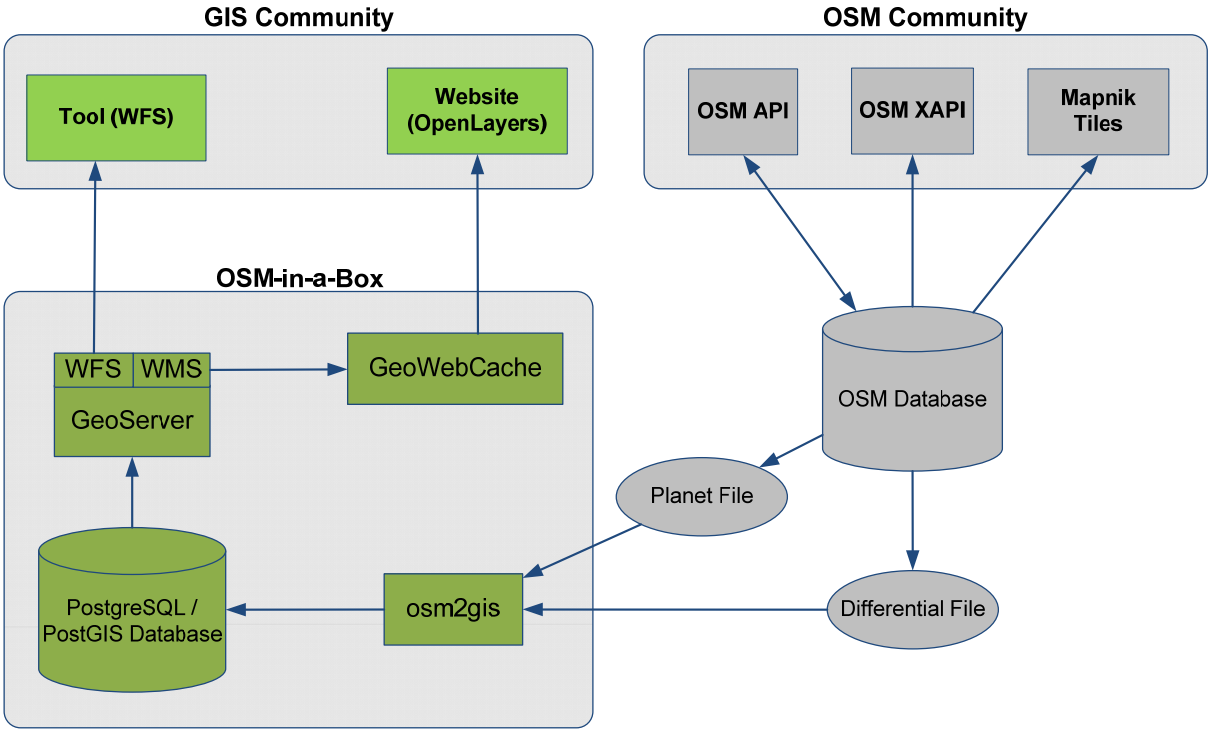


Abbildung 9: osm2gis Projektstruktur des Vorgängerprojektes.

Mit einem Mapping-File können die zu importierenden OSM-Daten in ein beliebiges Zielschema konvertiert werden. Dieses Zielschema muss vor dem erstem Import definiert sein, damit in einem Präprozess die Datenbankstruktur erstellt werden kann. Mit einer Konsistenzüberprüfungs-Funktionalität kann das Mapping-File auf Konsistenz zur Datenbank überprüft werden, damit Änderungen vom Importer bemerkt werden. Ebenso kann die Grafikkonfiguration des GeoServers auf Vollständigkeit zur bestehenden Datenbankstruktur überprüft werden um so die Konfigurationsarbeit zu vereinfachen.

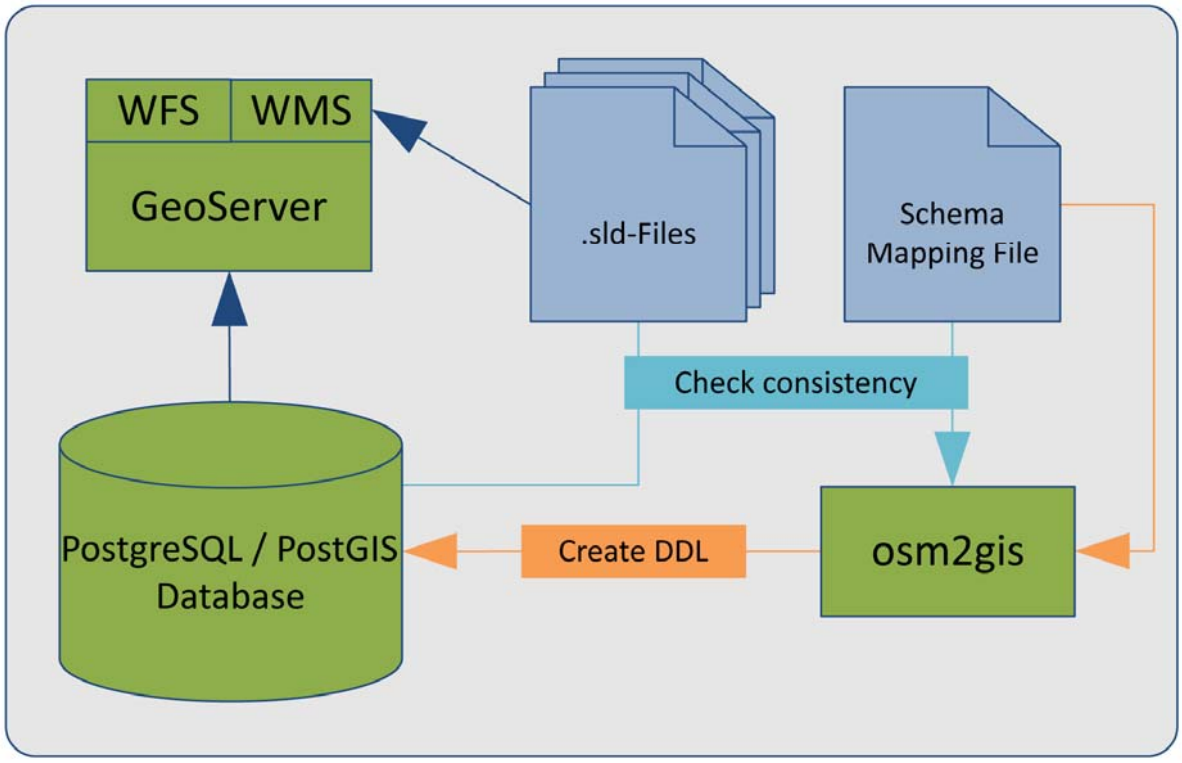


Abbildung 10:
Vorbereitungsphase in der einerseits anhand des Mapping-Files die Datenbankstruktur erzeugt wird und anderseits die .sld-Files (Grapfikkonfigurationsdateien) des GeoServers mit dem Mapping-File auf Konsistenz überprüft wird.

3.1.3 ZIEL

Die Abgabe einer robusten Software ist ein primäres Ziel dieser Bachelor Arbeit. Während den Erweiterungsarbeiten soll der osm2gis-Importer refaktorisiert und mittels TestCases stabiler gemacht werden. Folgende Erweiterungen und Teilarbeiten sollen gemacht werden:

3.1.3.1 ECHTES RELATION-HANDLING

Das aktuelle Relation-Handling (OSM Relation-Tag) verarbeitet lediglich Relations vom Typ *Multipolygon*. Diese Relations kommen insbesondere bei Flächen mit Löchern (Bsp. See mit Inseln, Wald mit Lichtung) vor, erweitern also die bestehende Möglichkeit, Flächen ausschliesslich mit Ways als geschlossene Polylinie zu erstellen.

Das Relation-Tag wird jedoch auch verwendet, um diverse andere OSM-Entitäten zusammen zu fassen. Beispielsweise referenziert eine Relation mit Typ *route* und Name *S9* alle Schienen und Bahnhöfe dieser S-Bahn. Ein Schienenabschnitt kann jedoch auch von einer anderen Relation mit Name *S15* referenziert werden, da beide Zuglinien diesen Abschnitt befahren. Um solche „echte“ Relationen in der Datenbank abspeichern zu können, muss der Import-Konverter um diese Funktionalität erweitert werden.

3.1.3.2 DIFFERENTIAL UPDATE

Um die Daten stets aktuell halten zu können, muss neben dem einmaligen Initial Import auch ein Update der Kartendaten möglich sein. OSM stellt hierfür sog. Differential-Files zu verfügung, die lediglich die geänderten Daten seit einem angegebenen Zeitpunkt enthalten. Die Differential-Update-Funktionalität war in der ersten Version 0.2 möglich. In folgenden Version 0.3 wurde sie jedoch deaktiviert, da sie mit der geänderten Funktionalität der Software nicht mehr kompatibel war.

Die Differential-Update-Funktionalität soll nun an die neue Software angepasst werden und auch das echte Relation-Handling unterstützen.

3.1.3.3 MOBILE WMS CLIENT

Für das Android OS soll eine Clientsoftware erstellt werden. Mit Hifle einer Library (muss noch evaluiert werden), soll der GeoServer via WMS-Abfragen das Kartenmaterial liefern welches der Client anzeigen soll. Rudimentäre Funktionen wie navigieren, zoomen sowie Suchanfragen tätigen soll der Client unterstützen.

3.1.3.4 UPDATE AUF GEOSERVER 2.0 & TOMCAT 6.0

Das OpenStreetMap-in-a-Box Projekt verwendet verschiedene Softwarekomponenten, für welche mitlerweile neue Versionen verfügbar sind. Speziell beim GeoServer hat sich einiges getan, weshalb die aktuelle GeoServer Version 2.0 ins OSM-in-a-Box Projekt aufgenommen werden soll. Anpassungen beim osm2gis ImportKonverter durch das GeoServer-Update müssen vorgenommen werden, da die Konsistenzprüfung mit der geänderten Ordnerstruktur im GeoServer nicht mehr kompatibel ist.

Funktionstests mit Tomcat 6.0 müssen ebenfalls vorgenommen werden.

3.1.3.5 ANPASSUNG KONSISTENZPRÜFUNG

Neben der Anpassung der Konsistenzprüfung wegen dem GeoServer-Update, müssen weitere Änderungen zur Unterstützung des neuen Mapping-Files für das Relation-Handling gemacht werden. Einerseits zur Überprüfung der Konsistenz zwischen Mapping-File und Datenbank, sowie der Konsistenz innerhalb des Mapping-Files selber.

3.1.3.6 GEOWEBCACHE, SHOWCASE

Die Showcase-Website soll an die getätigten Änderungen in dieser Bachelorarbeit angepasst werden. Dies beinhaltet hauptsächlich Arbeit an der Suchfunktion, damit die Relationen gefunden werden können.

Um die Ladezeiten des Showcase in einem Bereich wie andere Kartenanbierter wie Google-Maps, Openstreetmap zu haben, muss die GeoWebCache-Funktionalität beim GeoServer richtig einegstellt und aktiviert werden.

3.1.4 ANFORDERUNGEN IM DETAIL

Die Anforderungen an das Vorgängerprojekt sind im Dokument *Repository/root/tags/Server_V0.3/Dokumentation/08_Endabgabe/Gesamtdokumentation.docx Kapitel 3.1* enthalten.

3.1.4.1 FUNKTIONALE ANFORDERUNGEN

Funktionalität	Beschreibung/Bemerkungen	Priorität	Abhängigkeit
Relation-Handling	Mittels Mapping-File sollen auch für echte Relationen Mappings erstellt werden können.	Hoch	
DifferentialUpdate	Die DifferentialUpdate-Funktionalität soll mit den osm2gis-Erweiterungen komaptibel gemacht werden.	Hoch	Relation-Handling
Mobile WMS Client	Ein für Android OS tauglicher Client der via WMS Anfragen an den GeoServer tätigt soll erstellt werden	Mittel	

3.1.4.2 NICHT-FUNKTIONALE ANFORDERUNGEN

Lokale Datenbank

- Die OSM-Daten sollen weiterhin in eine Postgres Datenbank abgelegt werden. Um die Struktur der Daten effizient zu gestallten soll das Postgis Plugin von Postgres verwendet werden.
- Die Datenstruktur in der PostgreSQL Datenbank wird anhand des Mapping-Files vor dem Initial-Import erstellt. Vorteilhaft ist eine Datenstruktur nach dem Whitepaper von Jochen Topf[1].

Mapping-File

- Im Mapping-File soll die gwünschte Datenbankstruktur angegeben werden können mit:
  - Tabellen (Feld, Datentyp, Standardwert, Primary Key, References)
  - Views
- Soll die Datenstruktur via Mapping-File geändert werden, muss ein kompletter Datenimport erfolgen.
- Falls sinnvoll, müssen im Mapping-File geschütze Bereiche angegeben werden können, damit schon vorhandene Tabellen oder Views nicht überschrieben werden.

[1] *Repository/root/trunk/Dokumentation/13_Referenz_Literatur/osm-data-in-gis-formats-02.pdf*

- Es erfolgt eine Zuordnung von Tags die ein OSM-Entity haben muss und die entsprechende Zieltabelle
    - Für „echte“ Relationen erfolgt zusätzlich eine Angabe über die Verknüpfungstabellen

Konsistenzprüfung

- Die Grafikkonfiguration des GeoServers soll mit der im Mapping-File angegebenen Datenstruktur verglichen werden und Abweichungen sollen aufgezeigt werden können. Dies soll die Konfiguration des GeoServers übersichtlicher und einfacher gestallten.

GeoServer 2.0

- Bestehende Funktionalität soll durch den Update des GeoServers auf Version 2.0 nicht eingeschränkt werden.
- Für den GeoServer müssen weitere Styles konfiguriert werden, um die neuen Views für „echte“ Relationen anzeigen zu können.
- Mittels entsprechendem Mapping-File soll die im Vorgängerprojekt entstandene Datenbankstruktur erweitert werden.
- Mittels Aktivierung des GeoWebCache sollen die Antwortzeiten des GeoServers verbessert werden.

Auslieferung

- Die Installation soll möglichst einfach sein. Dies wird erreicht durch:
    - Installation wie bisher mittels Batchfile
    - Standardwerte für konfigurierbare Einstellungen
    - Standard Mapping-File

Integrationsarbeit & Website

- Anpassungen der Suchfunktion
- Migration der Website an einen definitiven Platz

3.1.4.3 USE CASES

3.1.4.3.1 USE CASE DIAGRAMM

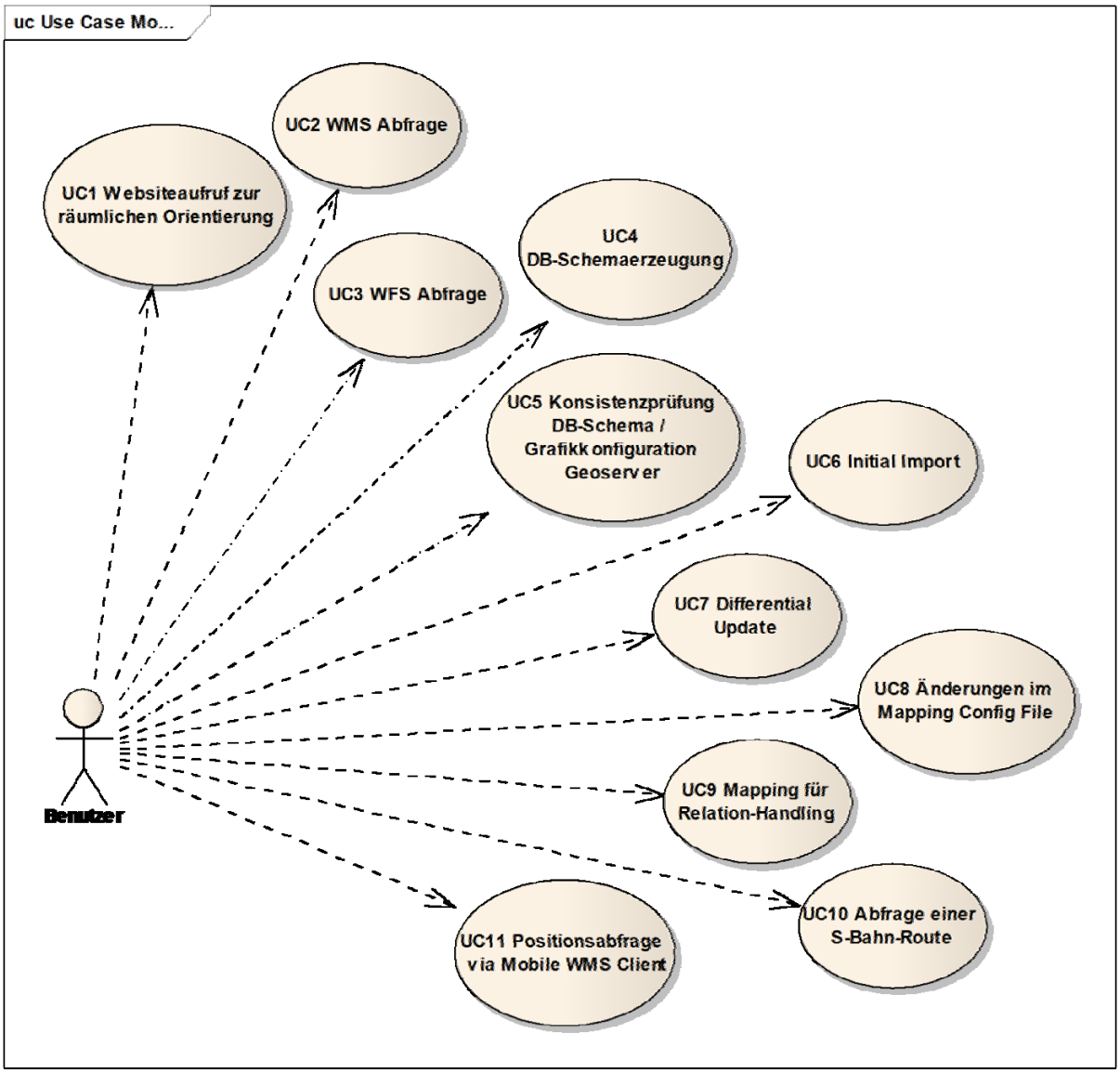


Abbildung 11: Use Case Diagramm mit den oberen acht Use Cases des Vorgängerprojektes und den drei neuen Use Cases.

3.1.4.3.2 USE CASES ÜBERSICHT

Als Benutzer ist der Server- und Websiteadministrator gemeint, welcher

- UC1 Websiteaufruf zur räumlichen Orientierung
- UC2 WMS Abfrage
- UC3 WFS Abfrage
- UC4 DB-Schemaerzeugung
- UC5 Konsistenzprüfung DB-Schema / Grafikkonfiguration Geoserver
- UC6 Initial Import
- UC7 Differential Update
- UC8 Änderungen im Mapping Config File

- UC9 Mapping für Relation-Handling
- UC10 Abfrage einer S-Bahn-Route
- UC11 Positionsabfrage via Mobile WMS Client

3.1.4.4 USE CASES DETAILIERT

3.1.4.4.1 UC1 WEBSITEAUFRUF ZUR RÄUMLICHEN ORIENTIERUNG

UMFANG	OpenStreetMap-in-a-Box
EBENE	Endbenutzer Ziel
PRIMÄRAKTEUR	Endbenutzer
STAKEHOLDER UND INTERESSEN	• Endbenutzer: o Der Endbenutzer möchte den Anfahrtsweg zur HSR einsehen.
VORBEDINGUNGEN	-
ERFOLGSGARANTIE	Der Endbenutzer konnte sich über den Anfahrtsweg an die HSR informieren.
STANDARDABLAUF	1. Der Endbenutzer ruft die Seite mit dem Anfahrtsplan auf. 2. Dem Endbenutzer wird die Anfahrtskarte dargestellt und er kann diese beliebig vergrössern, verkleinern sowie verschieben.
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• PostgreSQL • Geoserver • OpenLayers
HÄUFIGKEIT DES AUFTRETENS	Wird im Monat mehrmals pro Tag durchgeführt.

3.1.4.4.2 UC2 WMS ABFRAGE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer möchte via OpenLayers (eingebettet in eine Website) eine Kartenansicht von Rapperswil öffnen.
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	Öffnet OpenLayers (Eingebetet in eine Website) Client und wählt Rapperswil als Ort aus. Dem Benutzer wird den Kartenausschnitt von Rapperswil angezeigt.
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	OSM-in-a-Box OpenLayers
HÄUFIGKEIT DES AUFTRETENS	Sehr unregelmässig, je nach Aktivität 1-2x pro Woche

3.1.4.4.3 UC3 WFS ABFRAGE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer möchte die OSM Daten von Rapperswil und Umgebung in einem Vektorformat erhalten.
VORBEDINGUNGEN	Lauffähiger OpenStreetMap-in-a-Box Server
ERFOLGSGARANTIE	Der Benutzer konnte die vom Server geladenen Daten erhalten.
STANDARDABLAUF	Der Benutzer greift mittels eine Tool dass WFS unterstürzt auf den OSM-in-a-Box Server zu Der Benutzer bekommt die Vektordaten für den von ihm angegeben Bereich zurück.
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	OSM-in-a-Box WFS
HÄUFIGKEIT DES AUFTRETENS	Wenn im Einsatz sehr häufige Anfragen an den Server.

3.1.4.4.4 UC4 DB-SCHEMAERZEUGUNG

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer erstellt ein Mapping-Config File, so dass er selbst wählen kann welche Daten er importieren möchte und wie das Datenbankschema aussehen soll.
VORBEDINGUNGEN	• OSM-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht nach dem Initialimport in der Datenbank nur die Daten, welche er konfiguriert und gemappt hat.
STANDARDABLAUF	1. Öffnen des Mapping-Files, 2. Erstellen der Konfiguration der gewünschten Daten. 3. Starten von osm2gis mit Parameter (Initialimport).
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box • XML-Configfile
HÄUFIGKEIT DES AUFTRETENS	Vor dem ersten Datenimport.

3.1.4.4.5 UC5 KONSISTENZPRÜFUNG DB-SCHEMA / GRAFIKKONFIGURATION GEOSERVER

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Dem Benutzer soll eine Auswertung der Konsistenzprüfung des von ihm konfigurierten DB-Schemas und der Grafikkonfiguration des GeoServers geliefert werden.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht, welche vorkonfigurierten Daten noch nicht vom Geoserver gerendert werden können.
STANDARDABLAUF	1. Aufruf von osm2gis mit einem Startparameter. 2. Lesen der Auswertung.
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box
HÄUFIGKEIT DES AUFTRETENS	Wenn von Benutzer überprüft werden will, ob seine Grafikkonfigurationen im GeoServer vollständig sind.

3.1.4.4.6 UC6 INITIAL IMPORT

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o In der vom Benutzer erstellten Datenbank sollen die Tabellen die er Konfiguriert hat erstellt werden und das angegebene Planetfile importiert werden.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht in der Datenbank Tabellen die abgefüllt sind.
STANDARDABLAUF	1. Aufruf von osm2gis mit einem Startparameter. Angabe des ConfigFile. 2. Erfolg in DB überprüfen.
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box
HÄUFIGKEIT DES AUFTRETENS	Beim ersten aufsetzen des OSM-in-a-Box oder bei Änderungen des Configfile.

3.1.4.4.7 UC7 DIFFERENTIAL UPDATE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer sieht Änderungen und Erneuerungen die auf dem OSM Server gemacht werden, automatisch auf seinem System.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht die aktuellsten OSM Daten in seiner OSM-in-a-Box installation.
STANDARDABLAUF	1. Konfiguration eines Cronjobs
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box
HÄUFIGKEIT DES AUFTRETENS	Der Benutzer setzt einmal den Cronjob für das Differential Update auf.

3.1.4.4.8 UC8 ÄNDERUNG IM MAPPINGCONFIG FILE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer kann Zielschema und Mappingkonfiguration ändern.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht die Änderungen des Zielschema in der Datenbank sobald er einen Init Import ausführt. Die in den Mappingkonfigurationen konfigurierten Daten sind auf der Karte ersichtlich.
STANDARDABLAUF	1. Änderung des ConfiFiles 2. Ausführen eines Initial Import
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box
HÄUFIGKEIT DES AUFTRETENS	Beim ersten aufsetzen des Systems oder auf Wunsch des Benutzers.

3.1.4.4.9 UC9 MAPPING FÜR RELATION-HANDLING

UMFANG	OpenStreetMap-in-a-Box
EBENE	Benutzer Ziel
PRIMÄRAKTEUR	Benutzer
STAKEHOLDER UND INTERESSEN	• Benutzer: o Der Benutzer möchte „echte“ Relations in der DB haben (z.B. Relations S7 und S5 referenzieren dieselben Ways) und erstellt dafür die entsprechenden Mapping-Regeln im Mapping-Config File.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box
ERFOLGSGARANTIE	Der Benutzer sieht nach dem Initial Import, dass die gemappten Tabellen Werte enthalten.
STANDARDABLAUF	3. Änderung des Mapping-Files 4. Ausführen eines Initial Import
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• OSM-in-a-Box
HÄUFIGKEIT DES AUFTRETENS	Beim ersten aufsetzen des Systems oder auf Wunsch des Benutzers.

3.1.4.4.10 UC10 ABFRAGE EINER S-BAHN-ROUTE

UMFANG	OpenStreetMap-in-a-Box
EBENE	Endbenutzer Ziel
PRIMÄRAKTEUR	Endbenutzer
STAKEHOLDER UND INTERESSEN	• Endbenutzer: o Der Endbenutzer möchte die Route der S7 sehen und findet sie mittels Suchfunktion.
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box • Ein Relation Maping wurde beim Importieren der Daten angewendet
ERFOLGSGARANTIE	Als Resultate der Suche werden dem Benutzer die einzelnen Haltestellen sowie die Bahnschienen in der Resultatliste angezeigt. Mittels Mausklick auf einen Eintrag navigiert die Kartenanzeige auf den entsprechenden Eintrag.
STANDARDABLAUF	1. Der Endbenutzer ruft die Hauptseite von OSM-in-a-Box auf 2. Er gibt im Suchfeld den Begriff „S7“ ein und startet die Suche 3. Er wählt ein entsprechendes Suchresultat aus (z.B. „Route S7“) 4. Die Karte zoomt auf den Ausschnitt der Route der S7
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• PostgreSQL • Geoserver • OpenLayers
HÄUFIGKEIT DES AUFTRETENS	Mehrmals pro Monat.

3.1.4.4.11 UC11 POSITIONSABFRAGE VIA MOBILE WMS CLIENT

UMFANG	OpenStreetMap-in-a-Box
EBENE	Endbenutzer Ziel
PRIMÄRAKTEUR	Endbenutzer
STAKEHOLDER UND INTERESSEN	• Endbenutzer: o Der Endbenutzer möchte seine Position auf der Karte seines Android Handies ansehen
VORBEDINGUNGEN	• Lauffähiges OpenStreetMap-in-a-Box • Das Handy hat via Internet Konektivität zum GeoServer • Das Handy kann anhand GPS die eigene Position ermitteln
ERFOLGSGARANTIE	Anhand der eigenen Position wird die umliegende Karte vom GeoServer angefordert. Der Endbenutzer sieht seine Position zentriert auf der Karte.
STANDARDABLAUF	1. Der Endbenutzer startet seine WMS Client auf dem Handy 2. Ohne weitere Eingabe zoomt die Karte auf seine aktuelle Position
ERWEITERUNGEN	Keine
LISTE DER TECHNIK- UND DATENVARIATIONEN	• Android OS • GeoServer • Geeignete WMS-Lib
HÄUFIGKEIT DES AUFTRETENS	Mehrmals pro Monat.

3.2 DESIGN

3.2.1 INTRODUCTION

3.2.1.1 PURPOSE

This document gives the reader an overview over the design of this software. It's highly recommend to read this document if you plan to extend or renew the software that was built in the prior bachelor and semester thesis and updated to the current version 1.0 in this bachelor thesis.

3.2.1.2 SCOPE

The scope is limited to the duration of this project. Some chapters or parts of them were copied or updated from the prior bachelor / semester thesis (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**) to give the reader a full view of the current state. Changes in future iterations are possible.

3.2.1.3 VERSION 1.0 NOTES

Since the OSM-in-a-Box project is part of a two bachelor-, a semester- and a diploma thesis, some version notes are appropriate here:

Version	Thesis & Authors	Notes
Server 0.2	Bachelor thesis, Spring 2009 by Fabio Renggli, Mike Huber and Roland Hof	First release of the OSM-in-a-Box software.
Mobile 0.1	Diploma thesis by Marco Busarello	Based on Server V0.2, work on the Map-Styles and the Showcase website was done as well as an Android OS client application. The Mobile-Software is released separately.
Server 0.3	Semester thesis, Autumn 2009 by Andreas Meier and Joram Zimmermann	Based on Server V0.2, work on the osm2gis application was done. The updated Style-Files and website from Mobile V0.1 was included in this release.
Server 1.0	Bachelor thesis, Spring 2010 by Andreas Meier and Joram Zimmermann	The current work on the server part brings the software to version 1.0 when released in summer 2010.

There are several goals to achieve in this bachelor thesis. One is to have a stable Server Version 1.0 ready to be released. This includes several bug fixes as well as Unit- and BlackBox-Tests for the current and any continuous work. Changes are the support of real database relations (table – join table – table) based on OSM-Relations (the prior OSM-in-a-Box version only supported relations used for multipolygon-area). The DifferentialUpdate functionality needs to be greatly updated to support the user-defined configuration of the database scheme and the mapping rules which were added in the last semester thesis as well as the new relation handling. The relation handling support also requires an update in the consistency-check between the Schema Mapping File and the database.

3.2.1.4 DEFINITIONS AND SHORTCUTS

Term	Explanation
OSM	The Project OpenStreetMap. See http://www.openstreetmap.org
Node / Point	Terms are equal and define a Data type in OSM. See also http://wiki.openstreetmap.org/wiki/Data_Primitives
Way	OSM primitive data type : http://wiki.openstreetmap.org/wiki/Data_Primitives
Area	OSM Relation with type=multipolygon are converted into Area objects in osm2gis
Relation	OSM primitive data type : http://wiki.openstreetmap.org/wiki/Data_Primitives Note: In version 2.0 only multipolygon-type relations (areas) are supported.
Geom	Database column used to store Geospatial Information on PostGis.
WMS	*W*eb *M*ap *S*ervice: creates the tiles on GeoServer
WFS	Web Feature Service: Standardized interface on GeoServer which return raw vector data
OpenLayers	OpenSource program to display map tiles in a web browser. http://openlayers.org/
Planet-File	Weekly extract of the OSM database in xml format

3.2.1.5 REFERENCES

Library	Description	Source
Quartz	Used for scheduling Differential Updates in osm2gis	http://www.quartz-scheduler.org/
Log4j	Logging Library used in osm2gis	http://logging.apache.org/log4j/
GeoServer	MapServer to publish imported map data	http://geoserver.org

3.2.1.6 OVERVIEW

This document gives the reader an overview of the design of the osm2gis software. It includes the following elements:

- Physical and logical architecture
- Used configuration files
- Database structure

3.2.2 PHYSICAL ARCHITECTURE

The physical architecture of OSM-in-a-Box can best 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, can read the imported data and create tiles that can be seen in OpenLayers or you can do WMS or WFS requests.

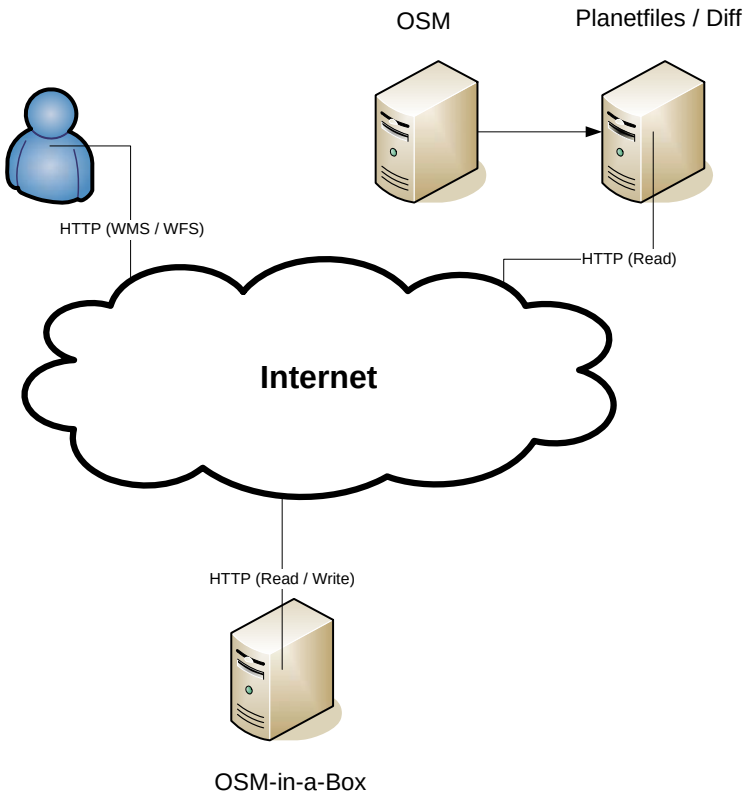


Figure 6: Physical architecture

3.2.3 LOGICAL ARCHITECTURE

3.2.3.1 OVERVIEW

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

Figure 7: Rough overview of the package structure

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

The *updating* package contains functionality for scheduling and executing an update job.

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

The *importing* 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 *importing*.

The *schemamapping* package is used when checking the database for consistency with the Schema Mapping File before an import occurs as well as the independent consistency check of the whole application (Schema Mapping File, database and GeoServer configuration).

3.2.3.1.1 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.

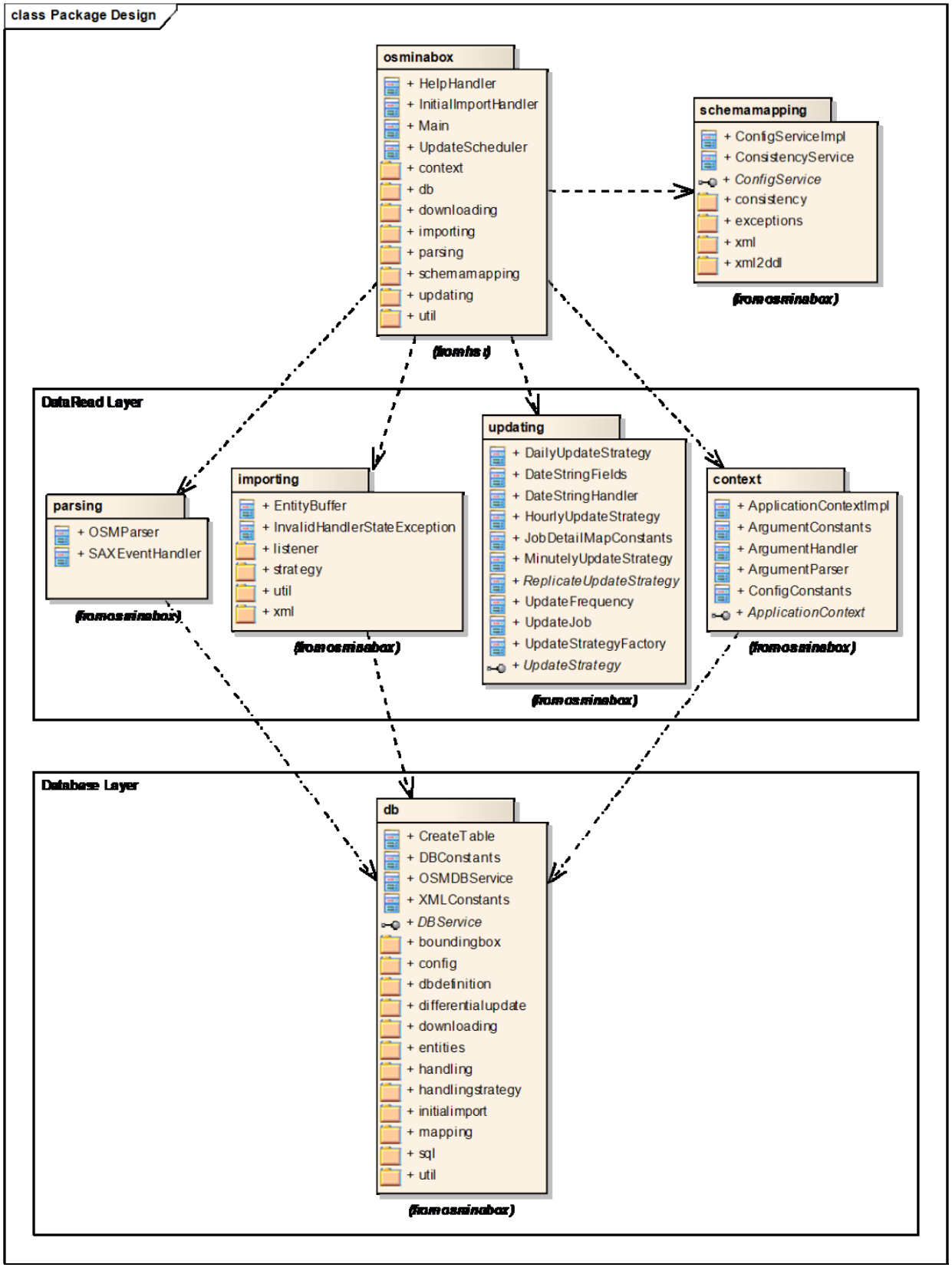


Figure 8: Package dependencies

3.2.3.2 GENERAL FUNCTIONALITY

The osm2gis application provides three 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.

CHECKING CONSISTENCY

The consistency check helps adjusting the different configuration files of the application to fit the user's needs. It creates a report about the consistency of the Schema Mapping File, the database and the GeoServer's graphic configuration.

How to generate the consistency report can be looked up in the user manual.

3.2.3.3 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 to specific handler classes. These provide the main chain of task execution for a specific functionality.

INVOLVED PACKAGES

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

IMPLEMENTATION APPLICATION INITIALIZATION

The main classes being part of the application initialization process:

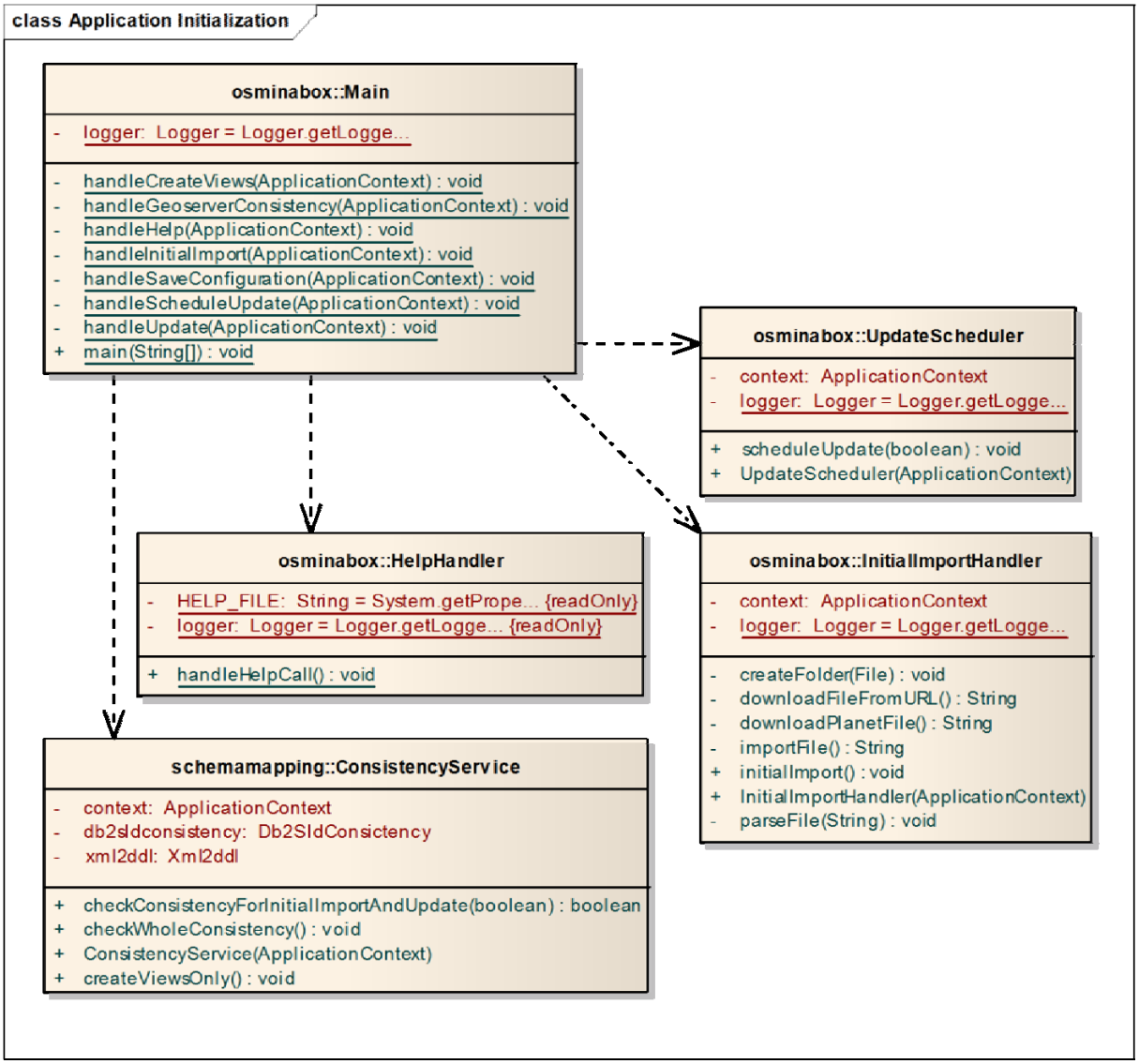


Figure 9: Application initialization

INITIALIZATION SEQUENCE

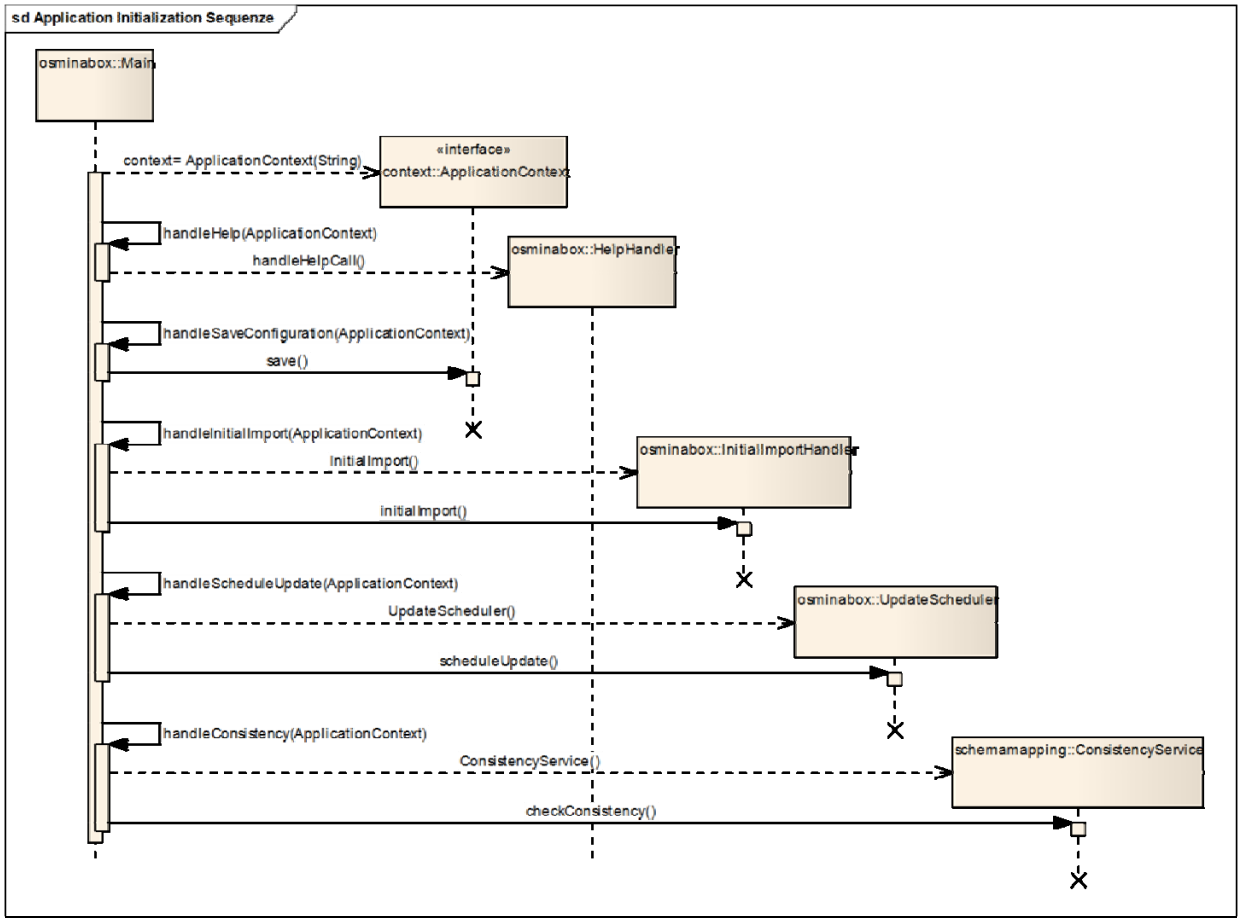


Figure 10: Application initialization sequence

Explanations

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

- InitialImportHandler
  Handles the initial import of OSM-Planet files or shape files. A shape file is basically a subset of a planet file.
- UpdateScheduler
  Schedules an update job to be executed only once or at a regularly time interval. Downloads and imports differential update files from OSM.
- HelpHandler
  Prints the 'man' page to the standard output.
- ConsistencyService
  Generates a report file about the consistency of all configurations (Schema Mapping File, database, GeoServer graphics configuration).

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

3.2.3.4 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
- Provide access to the Schema Mapping File via the ConfigService

These four 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
ch.hsr.osminabox.context	Contains the ApplicationContext class and some helper classes for argument parsing.

IMPLEMENTATION APPLICATIONCONTEXT

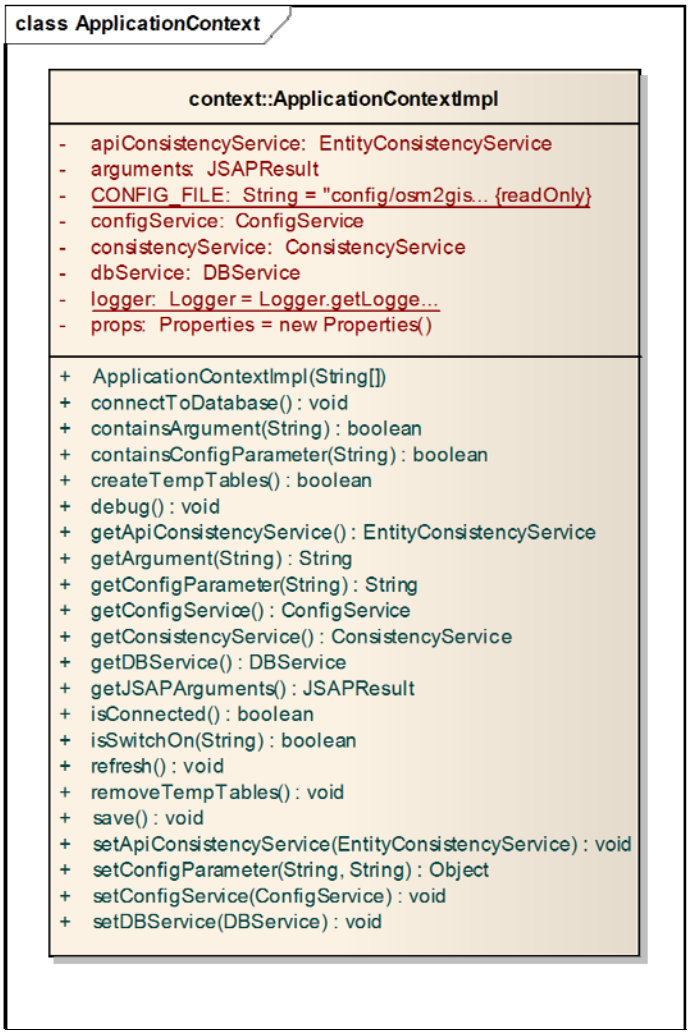


Figure 11: ApplicationContext

Explanation

The ApplicationContext provides methods to get and set configuration parameters. These can either be read from the configuration file or passed in as command line arguments when starting the software. With 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.

3.2.3.5 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.

3.2.3.5.1 BASIC PARSING ARCHITECTURE DECISION

Main Issue

To achieve a parsing process which is reliable and scales well, we needed 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. 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. Also a way which is closed can be used as an area (forest, lake, etc.). Relations are there to connect several elements on the map and give them a common meaning. One thing the relations are important for, within the application, is to connect multiple ways to one area. Such an area is a multipolygon with holes and / or multiple areas (exclaves). The other important meaning of relations is to hold information about real relations like a train route which references multiple railway-ways.

The problem is, that the different elements are stored separated 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 nodes, ways and even other relations. 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.

3.2.3.5.2 FUNCTIONALITY PARSING PROCESS

The Streaming API for XML (SAX) 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.

3.2.3.5.3 INVOLVED PACKAGES

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

3.2.3.5.4 IMPLEMENTATION XML HANDLERS

3.2.3.5.4.1 ROOT HANDLER

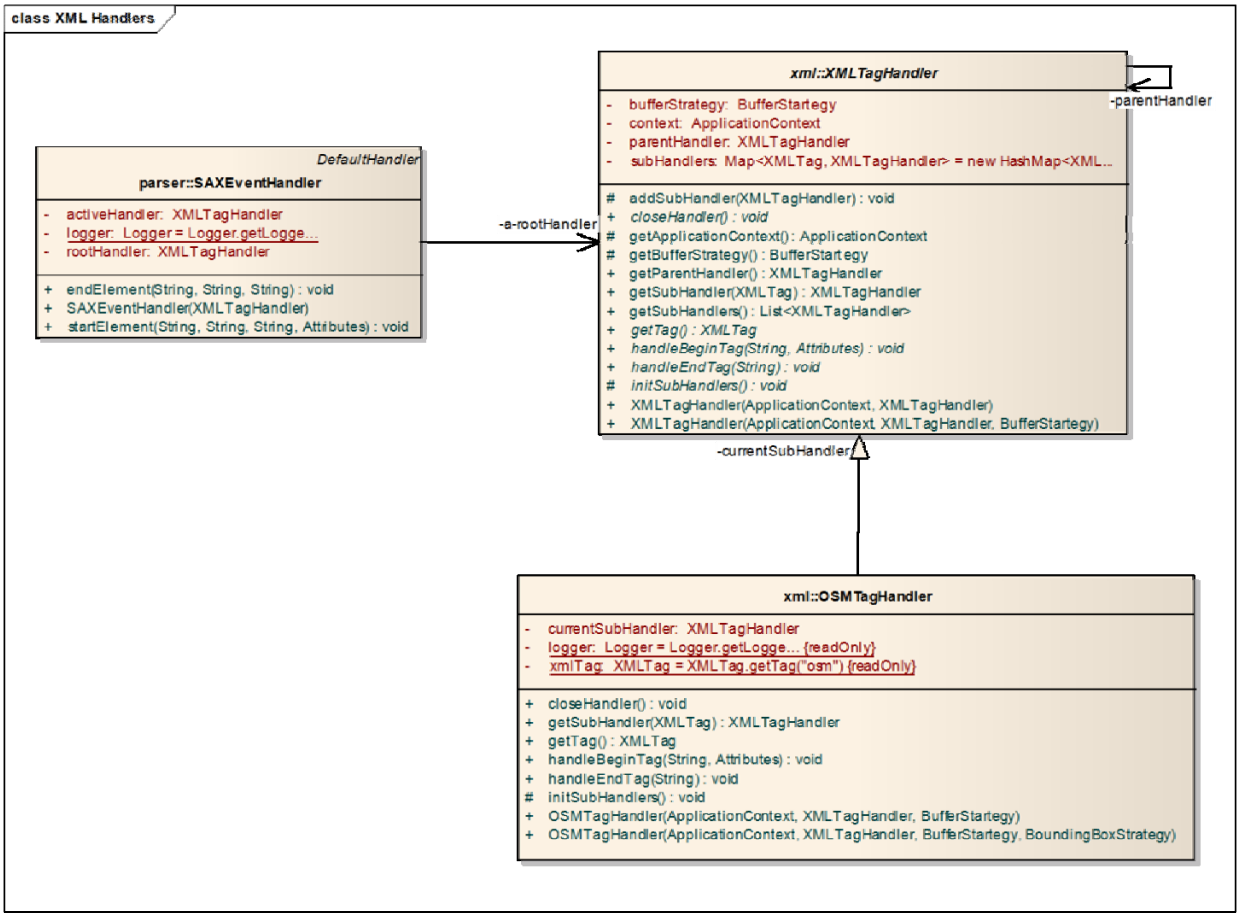


Figure 12: Root handler

Explanation

- *SAXEventHandler*
  The SAX 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 from OSM.

3.2.3.5.4.2 TAG HANDLER IMPLEMENTATIONS

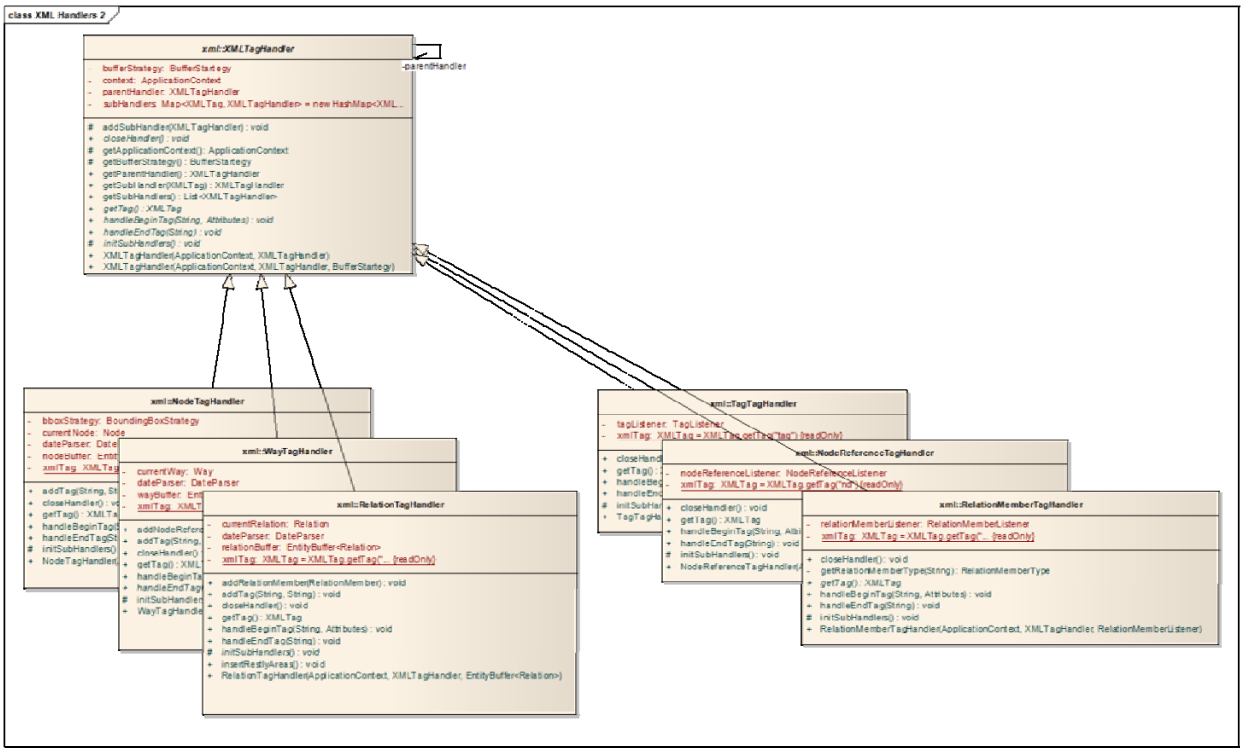


Figure 13: Tag Handler

Explanation

As said before, there is an implementation of an XMLTagHandler for every xml element that could occur 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. Can be a sub element of way.
- Member
  Represents a reference to a node, way or relation within a relation. Can be a sub element of a relation.

3.2.3.5.4.3 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 elements and parent elements. For example if a tag element occurs, the XMLTagHandlers knows his parent XMLTagHandler and passes the tag information to it, so the parent XMLTagHandler can decide what to do with it, depending on which xml element is the parent element.

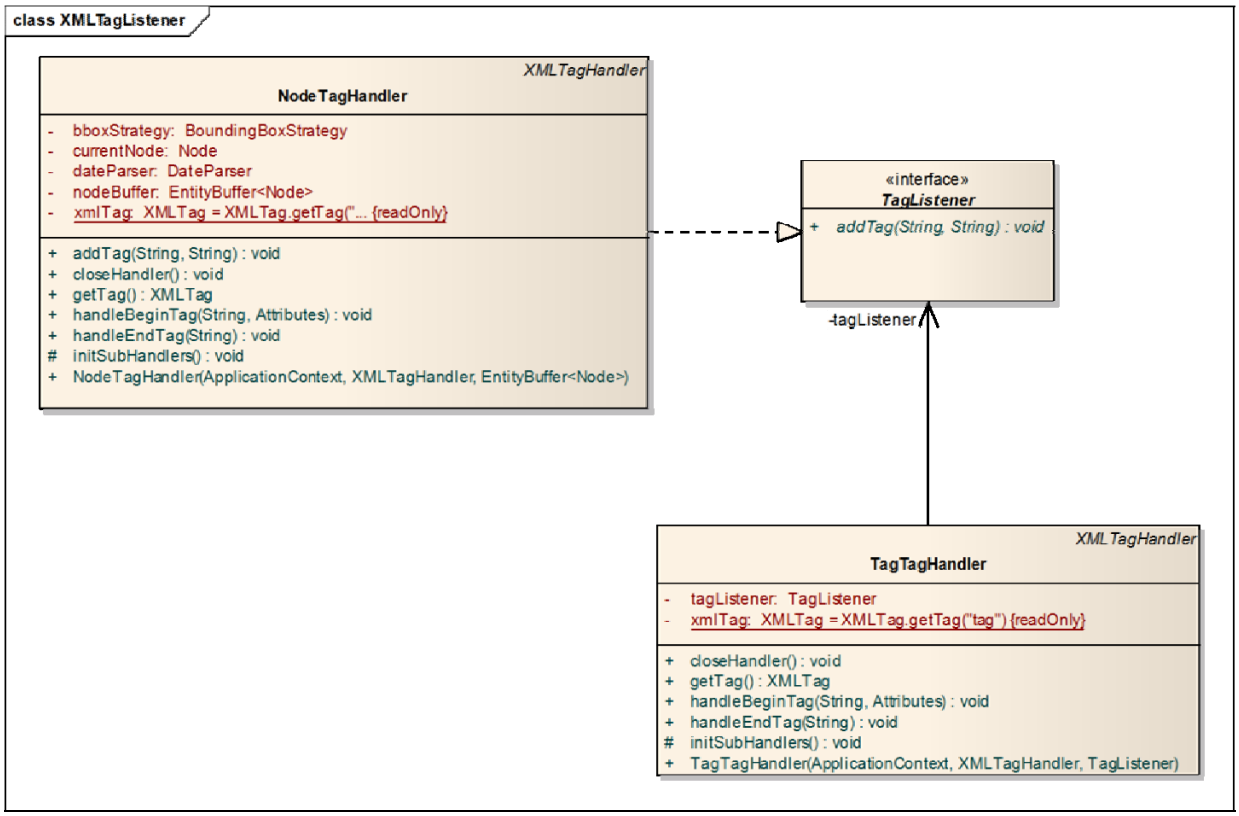


Figure 14: Handling Subtags

Explanation

Looking to the example with the tag described before, 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 so information will be passed through to the NodeTagHandler which adds the tag to the new node object.

3.2.3.5.5 SEQUENCE DIAGRAMM 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 occurrence in the xml file.

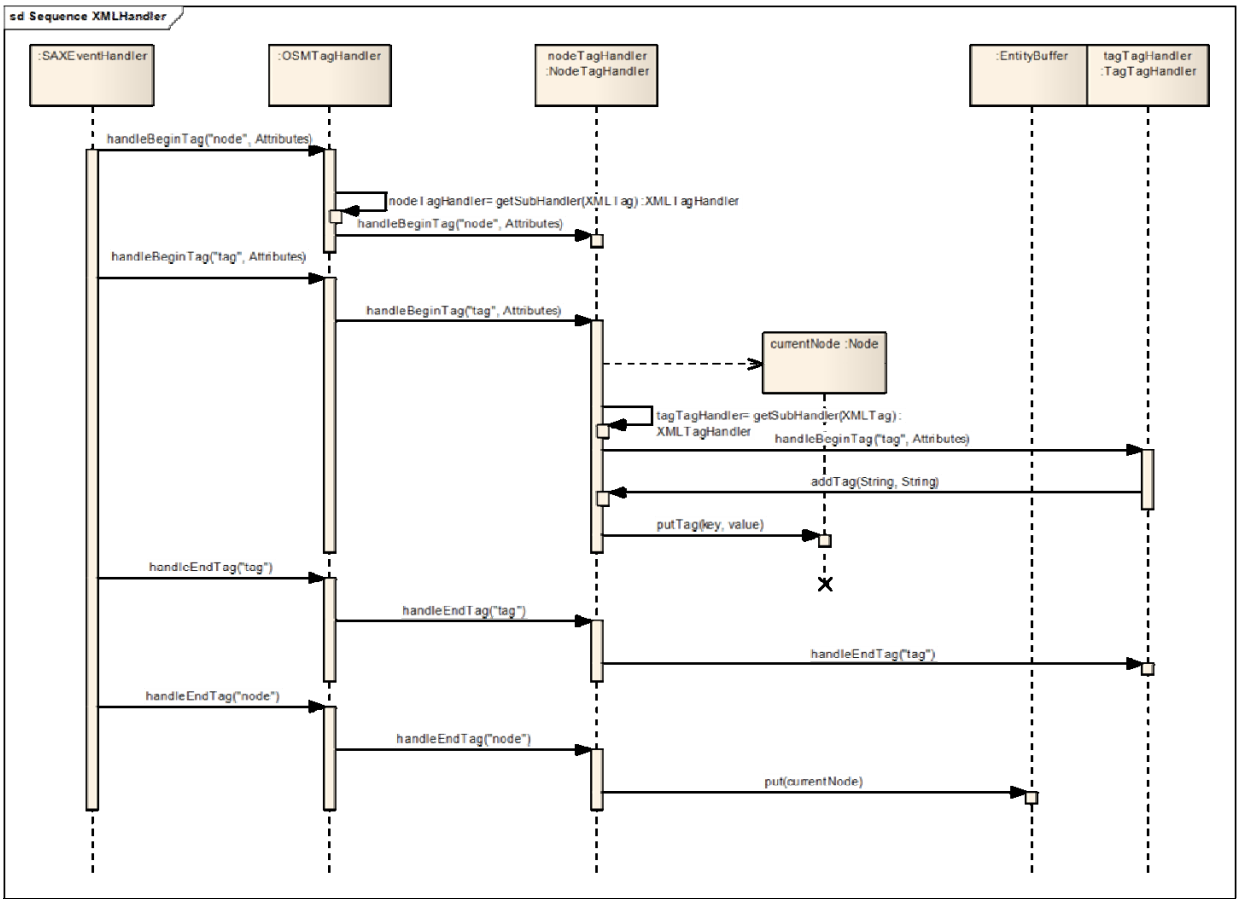


Figure 15: Tag Handling

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.

3.2.3.5.6 CREATING ENTITYS

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 node, way and relation an Entity will be created each time such an element occurs. This entity represents a node, way or relation from the OSM Data. A speciality is the OSM relation: if such a relation occurs, the RelationTagHandler decides, according to the already parsed tags belonging to this relation, if the relation should be treated as an area (type=multipolygon or type=boundary) or if it's a real entity to entity relation. So either an Area Object is created with the information from the relation or a Relation Object.

3.2.3.5.6.1 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 introduced. 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.

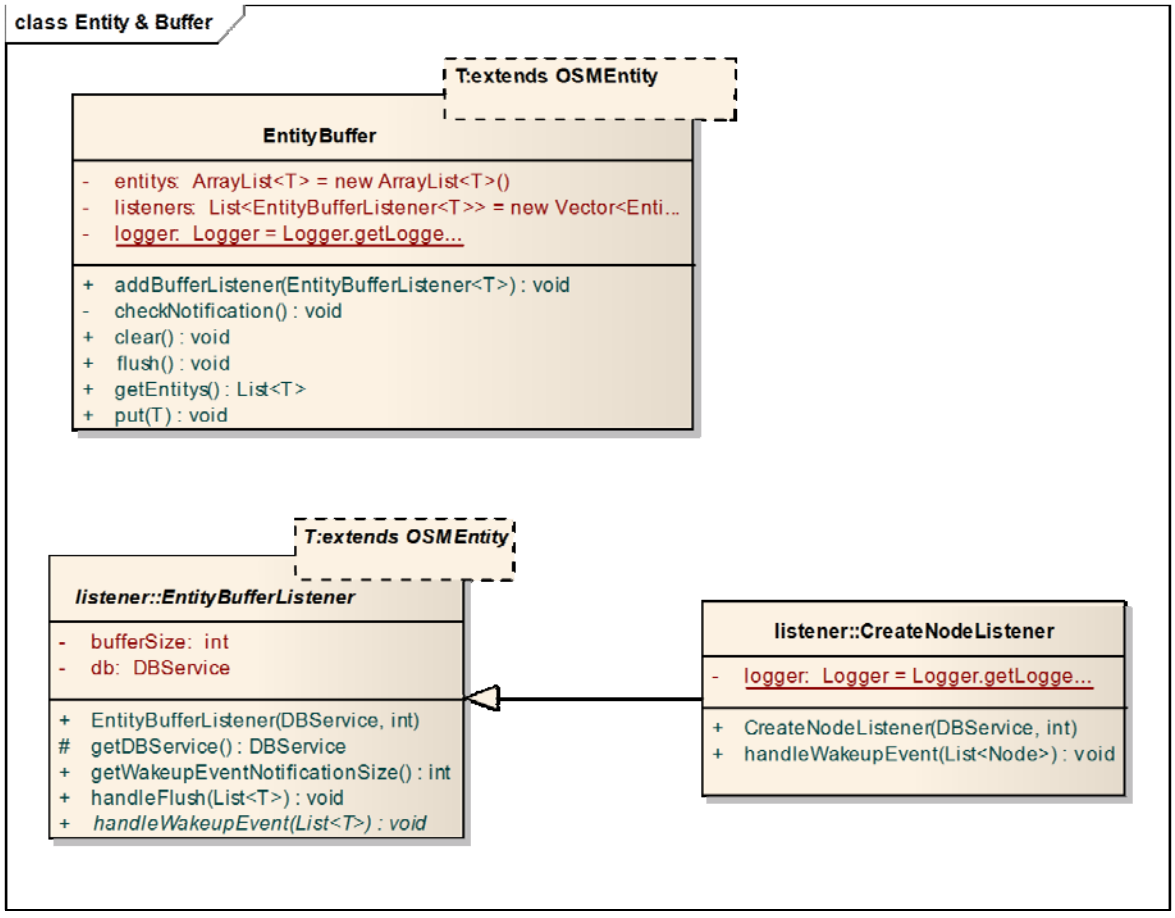


Figure 16: Buffer Strategy

Explanation

The diagram shows the EntityBuffer and the EntityBufferListener which can be registered at the EntityBuffer. The EntityBufferListener specifies the size of the buffer at which it will be notified. If the Buffer size is reached the EntityBuffer will notify all registered EntityBufferListener calling the handleWakeupEvent and pass its content through. The CreateNodeListener is an example of an EntityBufferListener implementation. It will pass the Nodes to the database layer calling the method for inserting nodes on the DBService. The database layer is described in another chapter. The RelationTagHandler has two EntityListeners attached one for areas and one for relations.

3.2.3.6 DATABASE: DATA MIGRATION

This chapter describes the migration of the data given to the database layer from *.*importing* package. Due to the size of this chapter is divided this into the following sub-chapter.

- General design of database layer
- BoundingBox
- Node Handling
  - InitialImport
  - DifferentialUpdate
- Way Handling
  - InitialImport
  - DifferentialUpdate
- Area Handling
  - InitialImport
  - DifferentialUpdate
- Relation Handling
  - InitialImport
  - DifferentialUpdate

NORMAL PROCEDURE OF DATA IMPORT/ DIFFERENTIAL UPDATE

The following flowchart shows the basic handling procedure of an initial import or a differential update.

1. **Initialize**
   a. The temp tables are created.
2. **Node handling**
   a. Every node in the xml-file is parsed and checked, whether or not it is inside the defined BoundingBox.
      i. If it is, the tags are checked if they map to one or many mapping entries in the Schema Mapping File.
         1. If mappings are found, the node is inserted in the corresponding table(s).
   b. Every parsed node is inserted in the node_temp table in any case (it might be referenced by ways).
3. **Way handling**
   a. Every way in the xml-file is parsed and checked, whether or not at least one referenced node lies within the defined BoundingBox.
      i. If one does, the tags are checked if they map to one or many mapping entries in the Schema Mapping File.
         1. If mappings are found, the way is inserted in the corresponding table(s).
   b. Every parsed way is inserted in the way_temp table in any case (it might be referenced by relations).
4. **Area handling #1**
   a. Every relation that is converted into an area object is checked, whether or not at least one way lies within the defined BoundingBox.
      i. If one does, the tags are checked if they map to one or many mapping entries in the Schema Mapping File.
         1. If mappings are found, the area is inserted into the corresponding table(s).
         2. All ways in the way_temp table, which are referenced by any inserted area, is marked as “used”.

5. **Relation handling**
    a. Every parsed relation (even if it was additionally converted into an area) is checked, whether its tags map to one or many mapping entries in the Schema Mapping File.
        i. If mappings are found, the relation is inserted in the corresponding table(s) and reference entries are inserted in the corresponding join table(s) to its referenced members.
        ii. All ways in the way_temp table, which are referenced by any inserted relation, is marked as “used”.
6. After all relations are parsed, every “used” way in the way_temp table by any relation is deleted.
7. **Area handling #2**
    a. Every remaining way in the way_temp table is checked, whether it is a single-closed-way (start- and endnode are equal) or not.
        i. A single-closed-way is transformed into an area-entity.
            1. Mapping entries for this area are looked up based on its tags.
            2. If one or multiple are found, the area is inserted in the corresponding table(s).
8. **Cleanup**
    a. All temp tables are deleted.

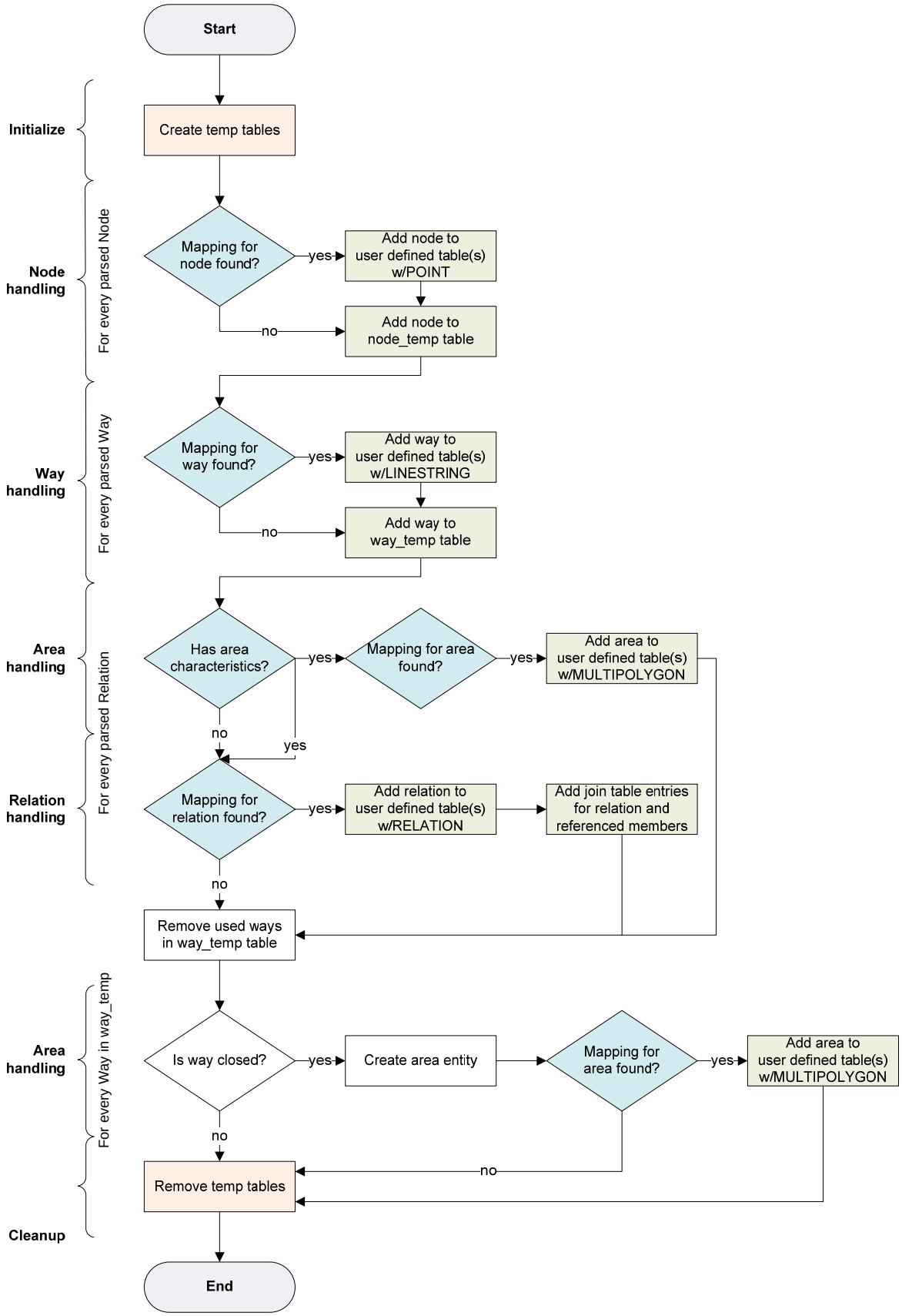


Figure 17: Flowchart of an Initial Import

3.2.3.7 GENERAL DESIGN OF DATABASE LAYER

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.

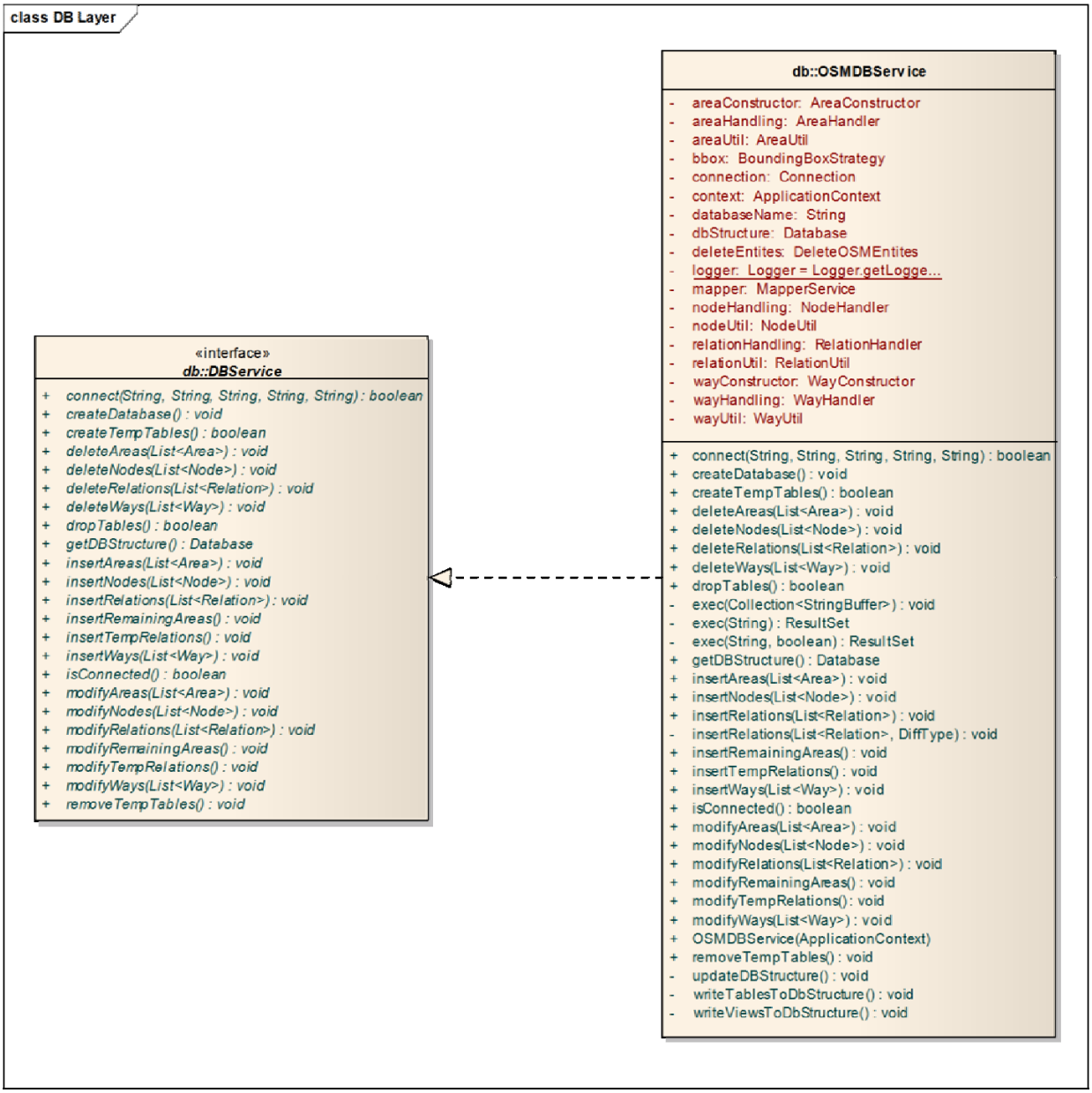


Figure 18: Database Layer

3.2.3.8 BOUNDINGBOX STRATEGY

In most cases there is a planet file which contains the OSM map information for the whole world or an extract for a country, but only a smaller area should 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.

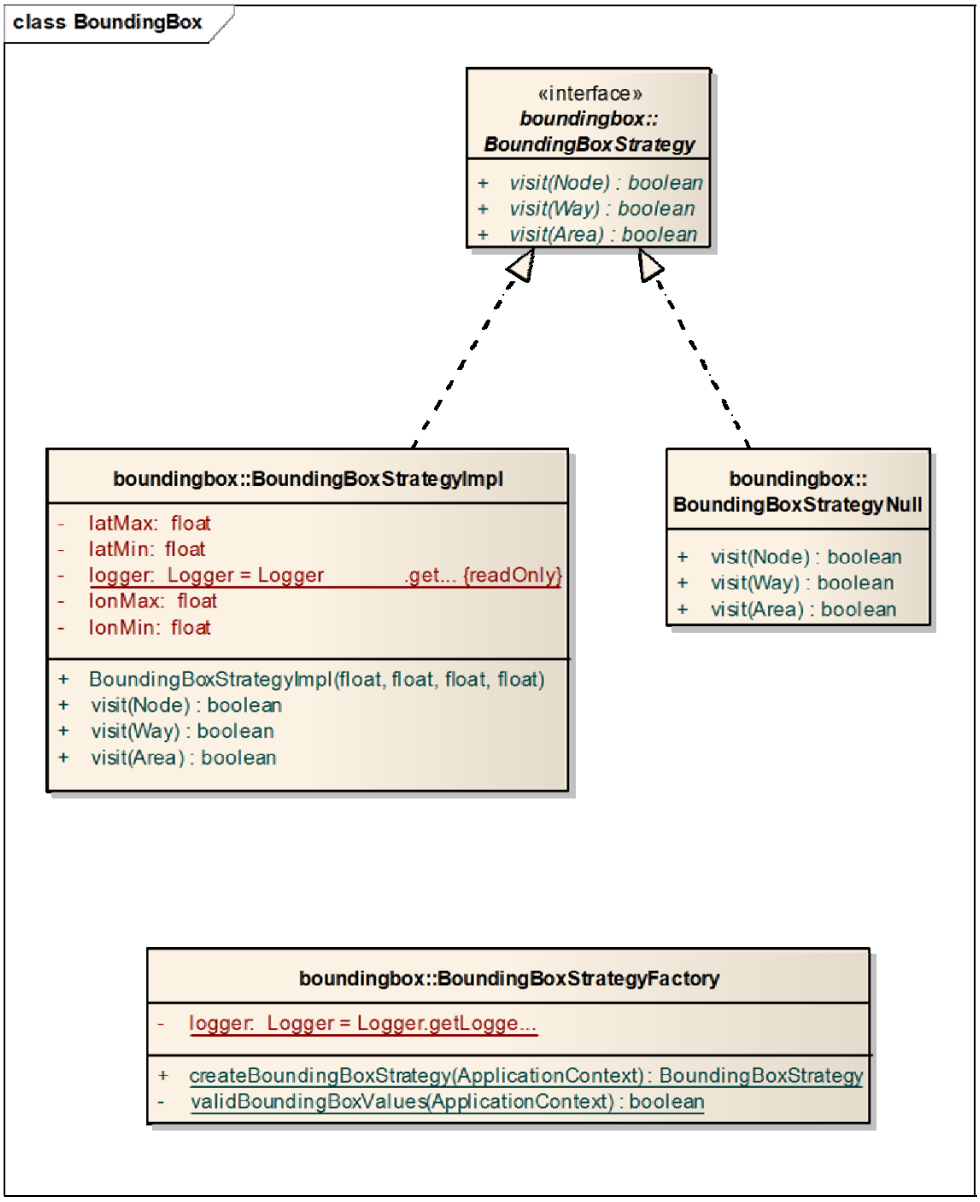


Figure 19: Boundingbox Strategy

Explanation

A BoundingBoxStrategy has three methods, a visit method foreither a node, way or area. 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.

3.2.3.8.1 BOUNDINGBOX FOR NODES

Determining whether a node lies within the BoundingBox or not is simple: the nodes lat / lon values must be greather respectively smaller than the BoundingBox's bounds.

3.2.3.8.2 BOUNDINGBOX FOR WAYS

Here the decision is needed, whether a way lies inside the BoundingBox when all of its nodes are inside the bounds or if it's sufficient when at least one node is inside the defined bounds. If all nodes must be inside the BoundingBox, all ways which are only partially inside the bounds are lost. Otherwise, ways which are partially outside will be imported. Since we want a complete dataset after importing, the second strategy is implemented.

3.2.3.8.3 BOUNDINGBOX FOR AREAS

An area lies within the BoundingBox, if at least one of its referenced way does. The same strategy as for ways is used and whether an area passes the visit method requires a visit for all ways which requires a visit for all nodes (or at least until the first entity passes).

3.2.3.9 NODE HANDLING

3.2.3.9.1 FUNCTIONALITY

The core functionality is to create SQL insert and update statements for a given list of nodes. This includes:

- Creation of SQL statements for the different database tables on initial import:
  Tables are completely user-defined but required columns are “geom” with the datatype “geometry” and “osm_id” of datatype “bigint”.
- Creation of SQL statements for the differential update:
  Depending on the modification, nodes need to be newly created, updated or deleted in any of the tables of geom type “point”.

3.2.3.9.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method insertNodes(..) and modifyNodes(..)
ch.hsr.osminabox.db.handlingstrategy	Node handling strategy
ch.hsr.osminabox.db.initialimport	Creation of INSERT sql statements
ch.hsr.osminabox.db.differentialupdate	Creation of UPDATE sql statements
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization and value conversion

3.2.3.9.3 IMPLEMENTATION NODE HANDLING

This diagram shows the implementation for node handling.

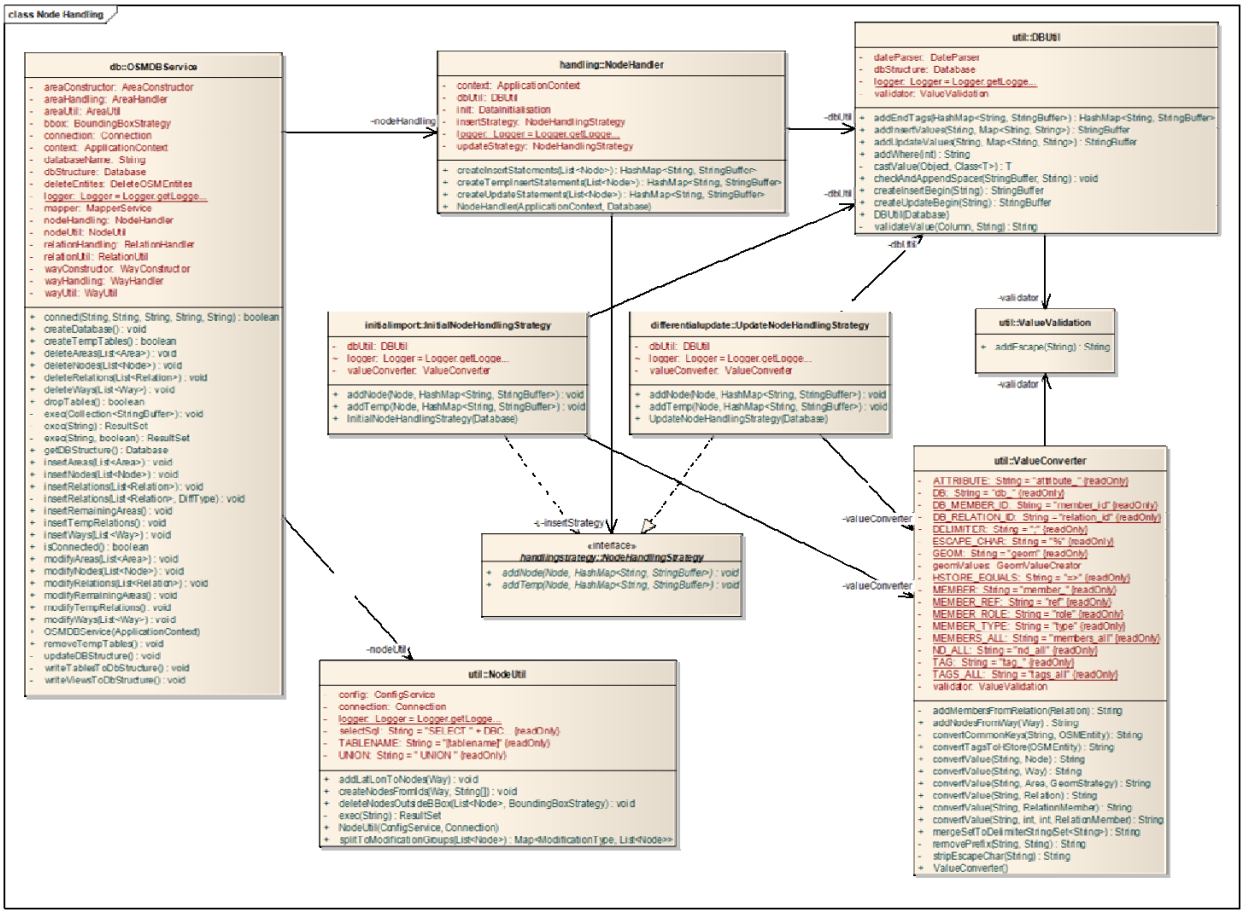


Figure 20: Implementation Node Handling

3.2.3.9.3.1 NODEHANDLING CLASS

Package: ch.hsr.osminabox.db.handling.NodeHandler

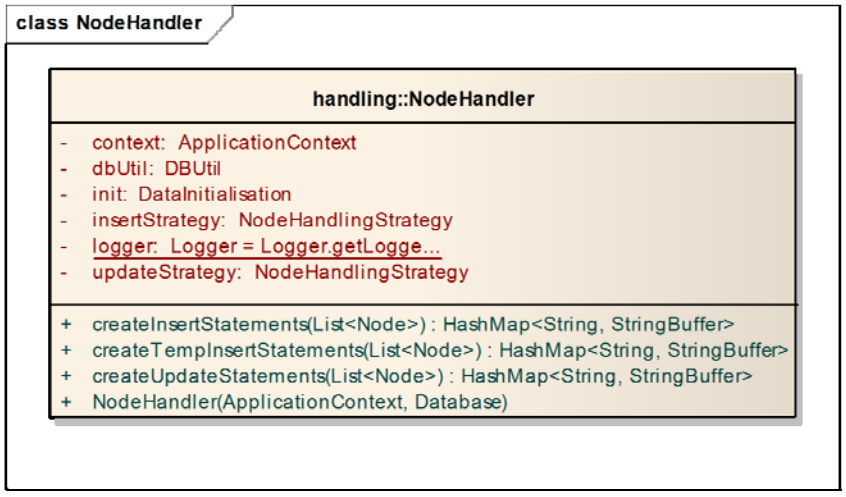


Figure 21: NodeHandler Class

The OSMDBService can call the methods needed for creating the sql statements to either insert or update nodes or for inserting nodes into the node_temp table.

The returned HashMap contains <String tablename, StringBuffer sql statement>. There is one SQL statement generated for all given nodes for initial import and multiple semicolon-separated update statements for differential updates.

3.2.3.9.3.2 SEQUENCE DIAGRAMM INITIAL IMPORT

This sequence diagram shows in a short manner the handling for a List of nodes on an initial import.

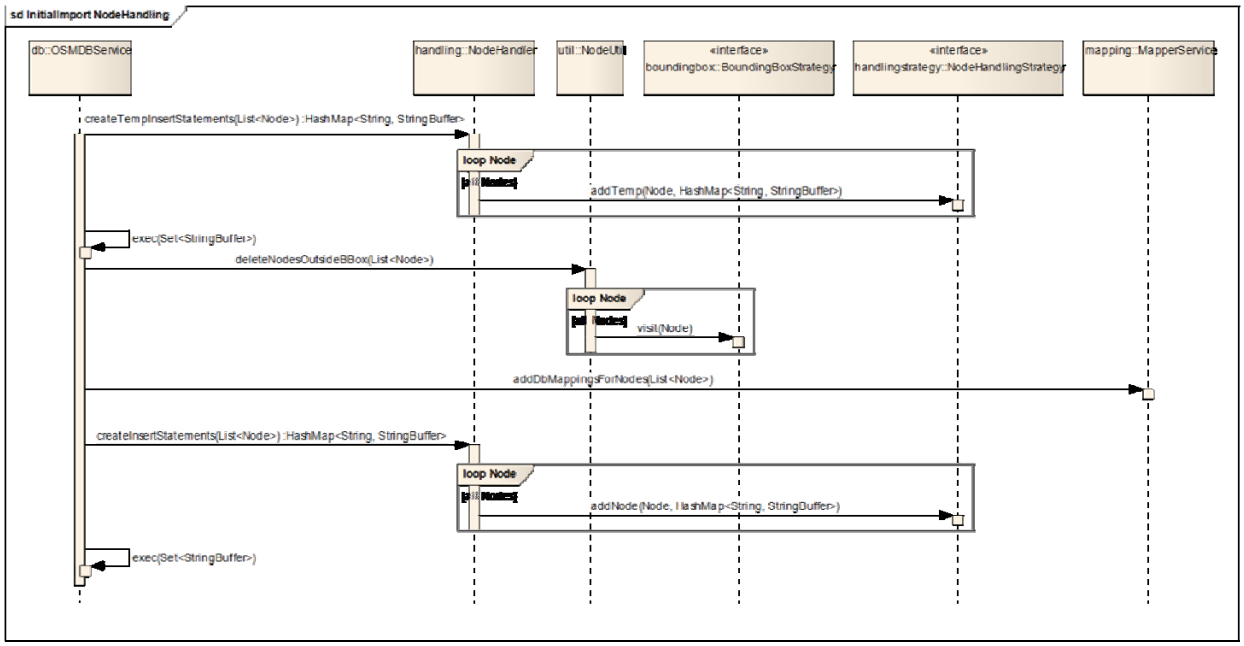


Figure 22: SD Initial Node Handling

3.2.3.9.3.3 FLOWCHART INITIAL IMPORT / DIFFERENTIAL UPDATE

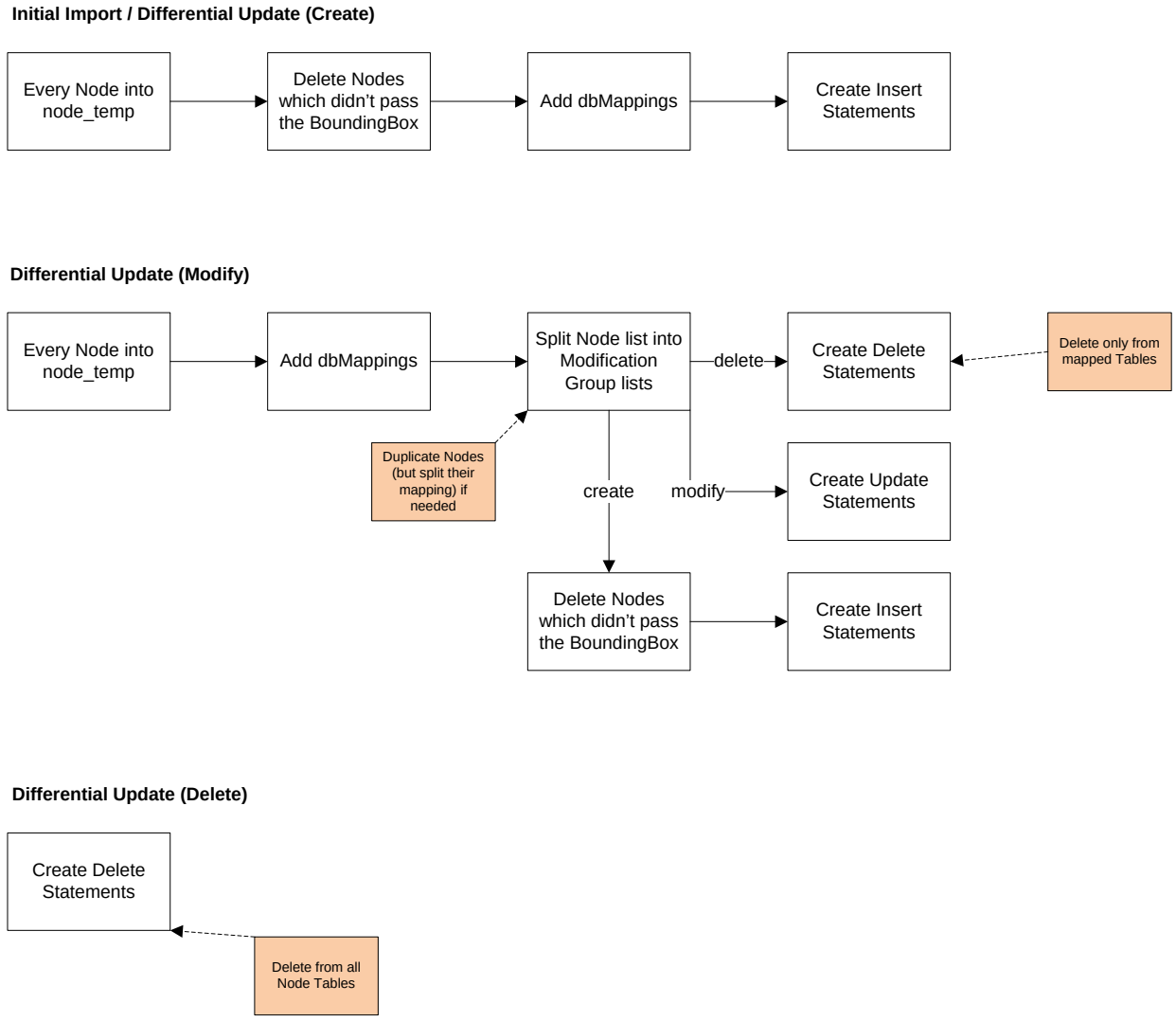


Figure 23: Flowchart for node handling on initial import / differential update

The general flow during an initial import and differential update can be seen in the above figure. A special note is needed to explain the modification groups: During a differential update, when nodes are parsed within the <modify> tag (which states that the nodes already exisit in the OSM DB but need to be modified), they can have a different impact on the local database aside a simple update. A node can change the database like this:

- Node has a new Tag which maps to a table it doesn't exist before -> INSERT node into mapped tables
- Node has changes but according to its tags it still maps to tables it is contained already -> UPDATE node in mapped tables
- Node doesn't map to tables in which it is contained -> DELETE node from tables

Of course any combination of the three modifications above are possible (Node needs to be deleted in one table, updated in another as well as newly inserted in a third one). If a node needs to be modified in more than one manner, the node is duplicated and added to the according modification list. Along with the node its dbMapping is split. Example: We have a database with two tables namely "shop" and "poi". In an earlier import a node was inserted into the "shop" table. Now we are running a differential update and the same node (osmId identifies this node) occurs within the <modify>-tag. To its original shop=supermarket tag (which mapped this node to the "shop" table in the first import) a new tag is added amenity=police. The MapperService finds two dbMappings for

this node, one belonging to the “shop” table and one to the “poi”. The point in the shop table needs to be UPDATED with the new node values but since it doesn’t exist in the “poi” table it needs to be INSERTED there. After the modification groups are created we have this node in the updateList and the insertList. But each node only contains the dbMapping for the specific table.

3.2.3.10 WAY HANDLING

3.2.3.10.1 FUNCTIONALITY

The core functionality is to create SQL Insert and Update statements of the given list of ways. This includes:

- Creation of SQL statement for the different database tables on initial and on differential import. Not all tables have the same attributes since they are completely user-defined.

3.2.3.10.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry Point with method insertWay(..) or modifyWay(..)
ch.hsr.osminabox.db.handlingstrategy	WayHandling strategy
ch.hsr.osminabox.db.initialimport	Start of the generation of the SQL scripts
ch.hsr.osminabox.db.differentialupdate	Creation of the initial way values
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.util	Creation of the differential values and retrieving node data
ch.hsr.osminabox.db	Util package used for way-checking functionalities and value conversion

3.2.3.10.3 IMPLEMENTATION WAY HANDLING

Classes that are used for way handling are shown below.

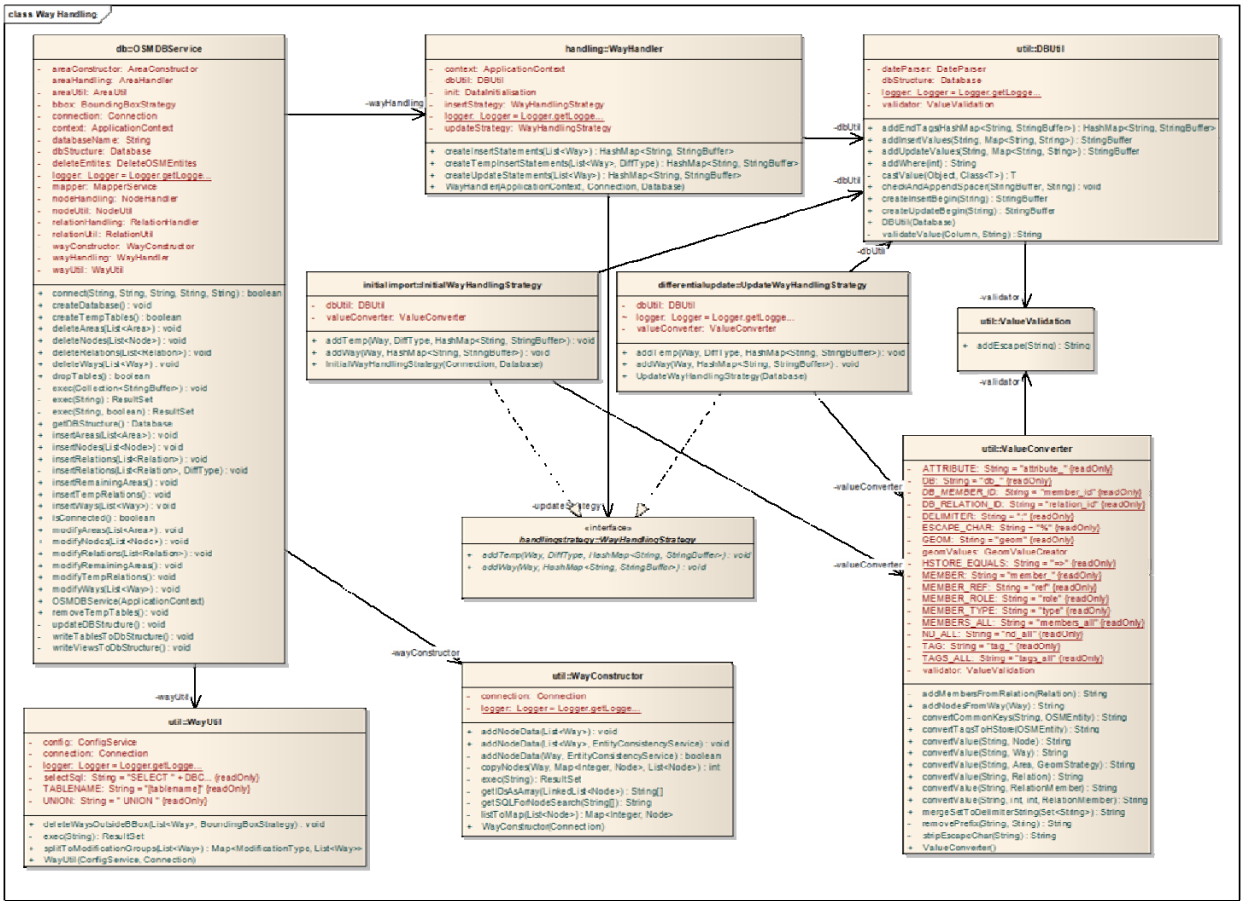


Figure 24: Implementation Way Handling

3.2.3.10.3.1 SEQUENCE DIAGRAM INITIAL IMPORT

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

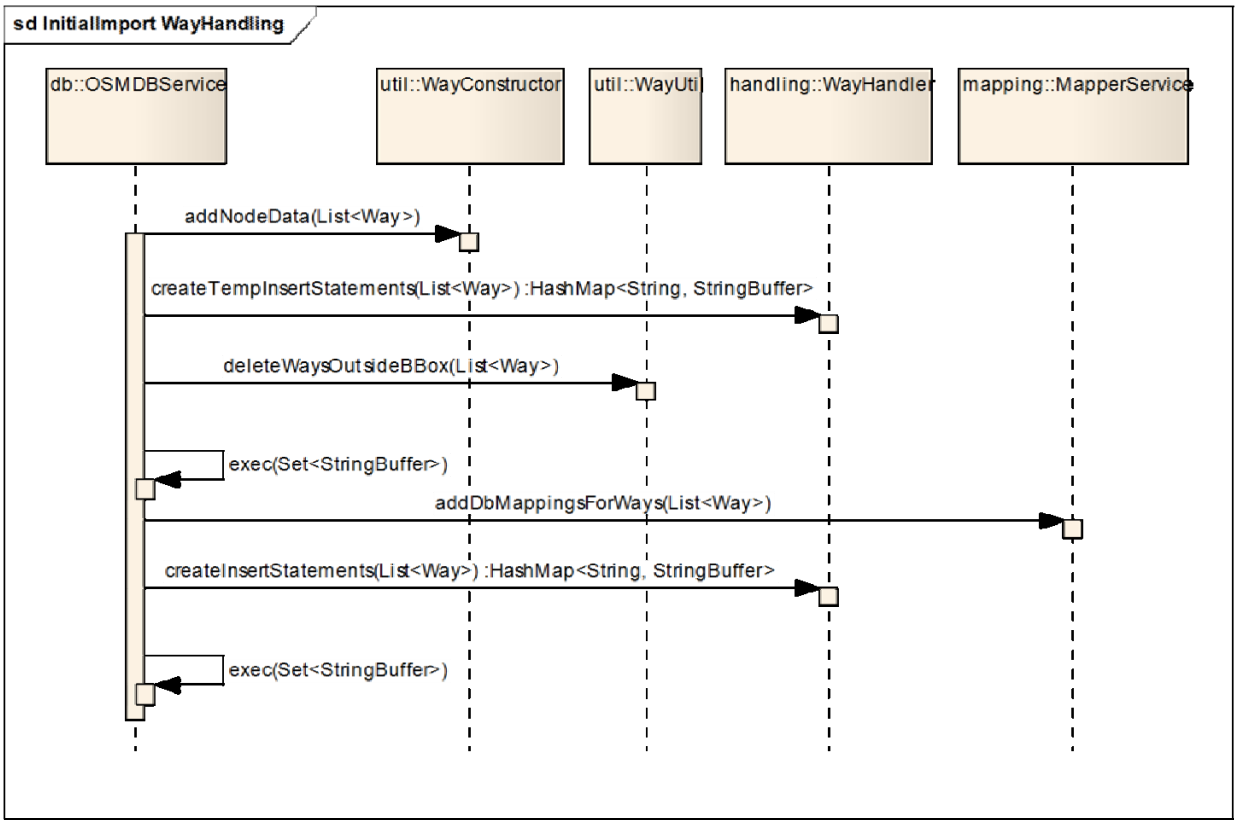


Figure 25: SD Implementation Way Handling

Explanations

Similar as with nodes, but prior of any sql statement creation, all nodes latitude / longitude values which are needed from a way must be retrieved from the databse node_temp table.

3.2.3.10.3.2 FLOWCHART INITIAL IMPORT / DIFFERENTIAL UPDATE

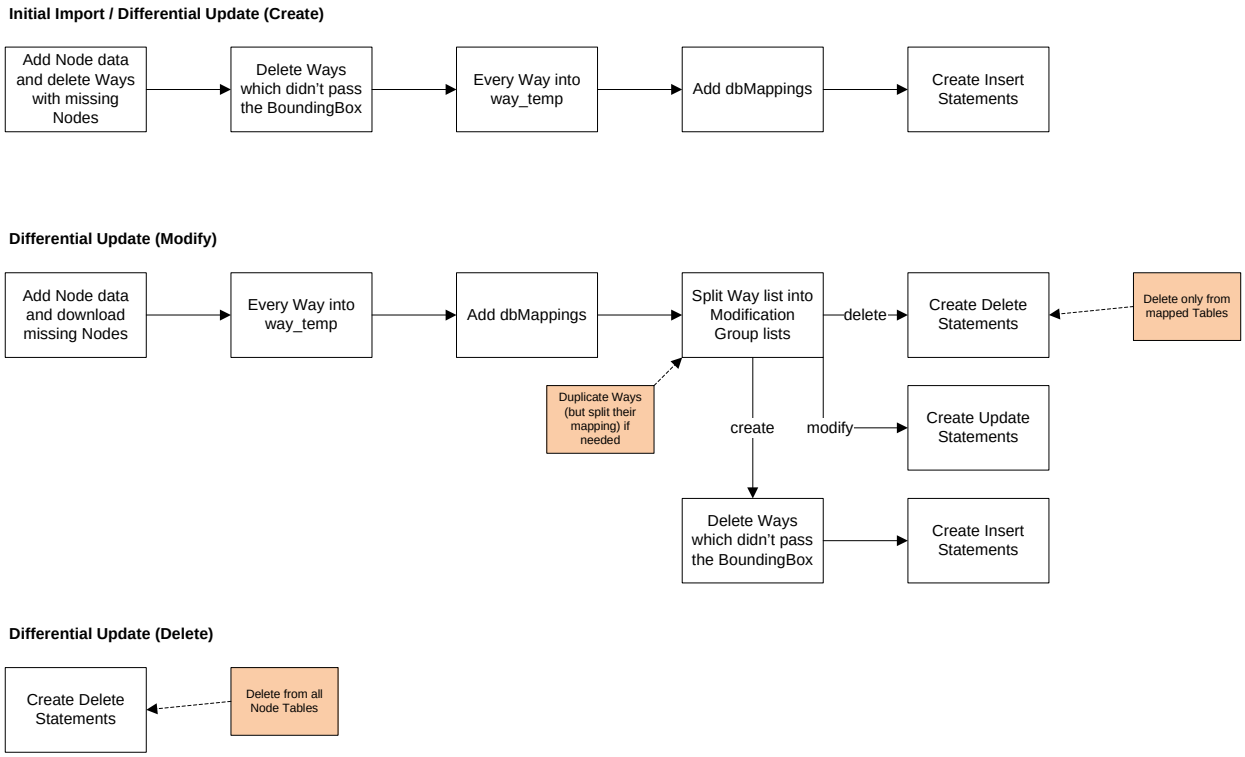


Figure 26: Flowchart for way handling on Initial import / differential update

The way handling on a differential update is basically the same like the node handling. The major difference is, that not necessarily every node used by a way is available inside the differential update file. Therefore it cannot be retrieved from the node_temp table. The EntityConsistencyService class downloads any missing node data via the OSM api so all needed values for a way is available.

3.2.3.11 AREA HANDLING

Area handling is one of the bigger parts of the osm2gis importer inside the database layer. Area is not an OpenStreetMap datatype like node and way, but a construct for handling all OSM entities which are supposed to be an area. OSM uses the relation with type = multipolygon to describe an area, but also single-closed-ways can be areas. And since a boundary of a town or country is a kind of an area as well, those relations are also converted into an area object inside the osm2gis importer.

3.2.3.11.1 FUNCTIONALITY

Here a list of the functionality implemented on area handling:

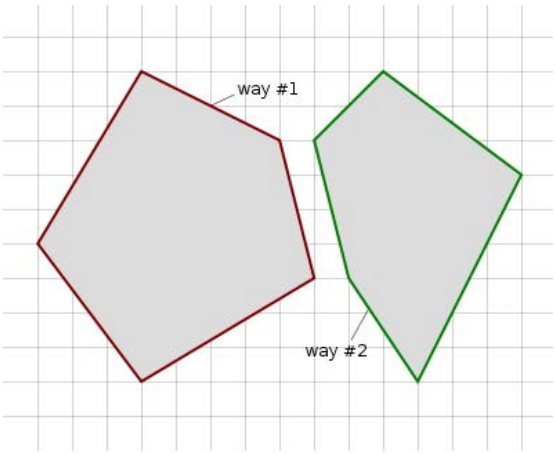
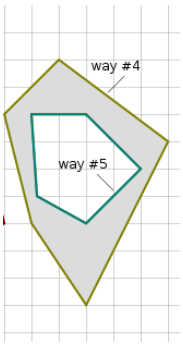
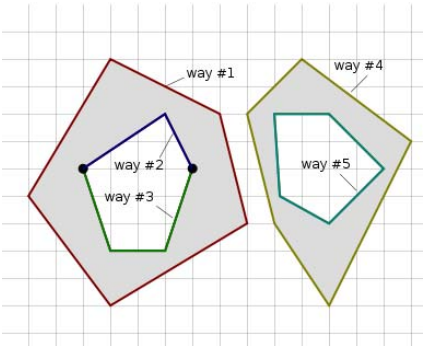
- Conversation from either a single-closed Way or a Relation into an area (multipolygon).
  - Intelligent Tag-inheriting from the source entity
- Creation of sql statements for initial imports and differential updates.
- Detection of area composition (needed to create correct sql statements).
- Special handling of areas with inner and outer ways / boundaries with enclaves and exclaves.
- Detection and combining ways to a single way.

3.2.3.11.2 INVOLVED PACKAGES

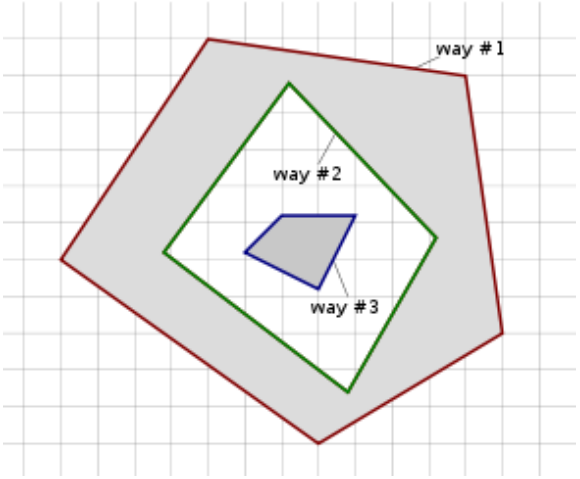
Packages	Explanation
ch.hsr.osminabox.db	Entry point with method insertArea(..) and modifyArea(..)
ch.hsr.osminabox.db.handlingstrategy	Area handling strategy
ch.hsr.osminabox.db.initialimport	Creation of the initial area values
ch.hsr.osminabox.db.differentialupdate	Creation of the update area values
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.util	Util package, used for constructing an area out of different components and for area composition detection

3.2.3.11.3 AREA COMPOSITION

The following types of area composition are possible[2]:

1.	The first Type is *only outer*. This means that the area has only ways that are outer boundaries of a polygon.	
2.	The second type has *one outer* and contains *several inner* ways. Normally this can be used to model a big area with holes.	
3.	The third type is the combination of type 1 and 2. This type can *have several inner and outer* ways. The specialty of the left area with 2 inner ways will be explained in the next chapter.	

[2] See http://wiki.openstreetmap.org/wiki/Relation:multipolygon

4.	The fourth type is a hole in a hole (in a hole…). Imagine a forest with a meadow. On the meadow are some more trees with a little pond inside. Inside Ways can be multipolygons of themselves. (But they have no own Relation). All outside Ways should have the forest tag, inside Ways either the meadow or water. Ring grouping is needed to see, which ring is nested in which other to create the correct geometry. This type is not supported by the OSM-in-a-Box software.	

3.2.3.11.4 COMPOSITION DETECTION

Depending on the composition of an area, different sql statements are needed to insert the area into the database. Simply because the sql statements are constructed in a different way for each of the three supported types (see above chapter), a detection algorithm is implemented within the AreaCompositionDetector class. The strategy pattern is used for the GeomUtil class to generate the corresponding sql statement according to the detected composition type of an Area.

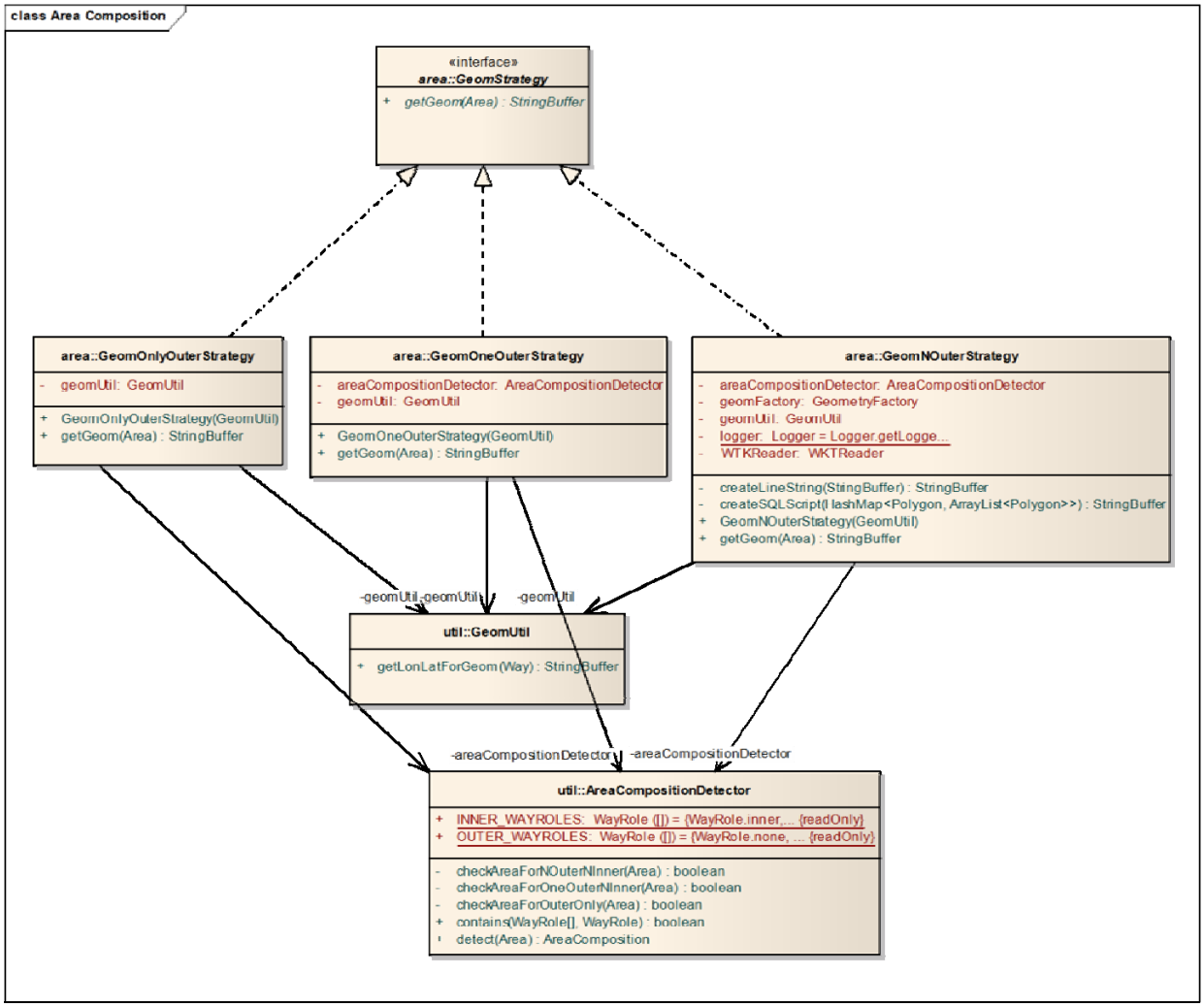


Figure 27: Area composition

3.2.3.11.5 COMBINING WAYS

To insert a complex geometry like an osm multipolygon into a database with PostGIS, the following steps need to be done to create the correct geometry values. Osm allows the boundary of an area to consist out of multiple ways which are connected (one's start point is another's endpoint). For PostGIS we need the coordinates of each Point in a list and therefore, if multiple ways exist in one so called *Ring*, they must be combined into a single Way.

The following, from OSM suggested algorithm, is implemented for combining Ways[3]:

Step	Description
RA-1	Assemble all ways that are members of the relation. Mark them as "unassigned", and reset the current ring count to 0.
RA-2	Take one unassigned way and mark it assigned to the current ring.
RA-3	If the current ring is closed (first node id == last node id): If the current ring is not a valid geometry (i.e. self-intersecting): Use backtracking to try other options of building this ring. If no other options exist, ring assignment has **failed**. If the current ring is a valid geometry, If there are any unassigned ways left, increase ring counter and go to RA-2. If there are no unassigned ways left ring assignment has **succeeded**
RA-4	If the current ring is not closed: Take current ring's end node and look for an unassigned way that starts or ends with this node If such a way is found, add this way to the ring and go to RA-3 If no such way is found, ring assignment has **failed**.

[3] See http://wiki.openstreetmap.org/wiki/Relation:multipolygon/Algorithm#Ring_Assignment

The above algorithm would combine Way #1 and #2 into a single Way for Figure 29. Leaving the new combined Way and Way#3 left for database insertion. A more complex (and more likely) example is Figure 29, Way #1, #2, #3 and #4 would be combined into a single Way etc..

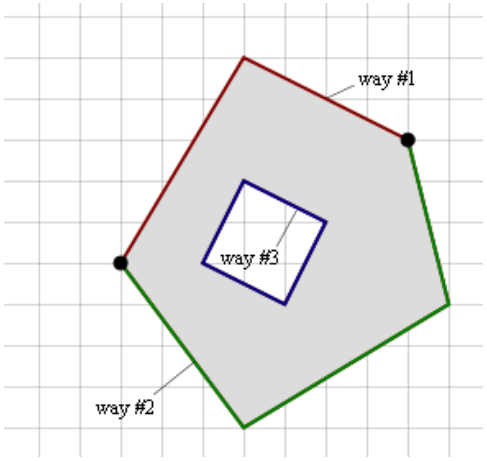


Figure 28: Way combining. Simple example

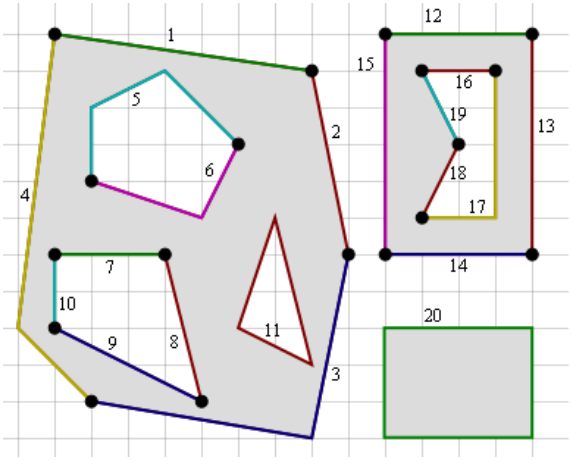


Figure 29: Way combing. More complex example

If way combining fails for an area, the area is ignored and not inserted in the database.

3.2.3.11.6 IMPLEMENTATION AREA HANDLING

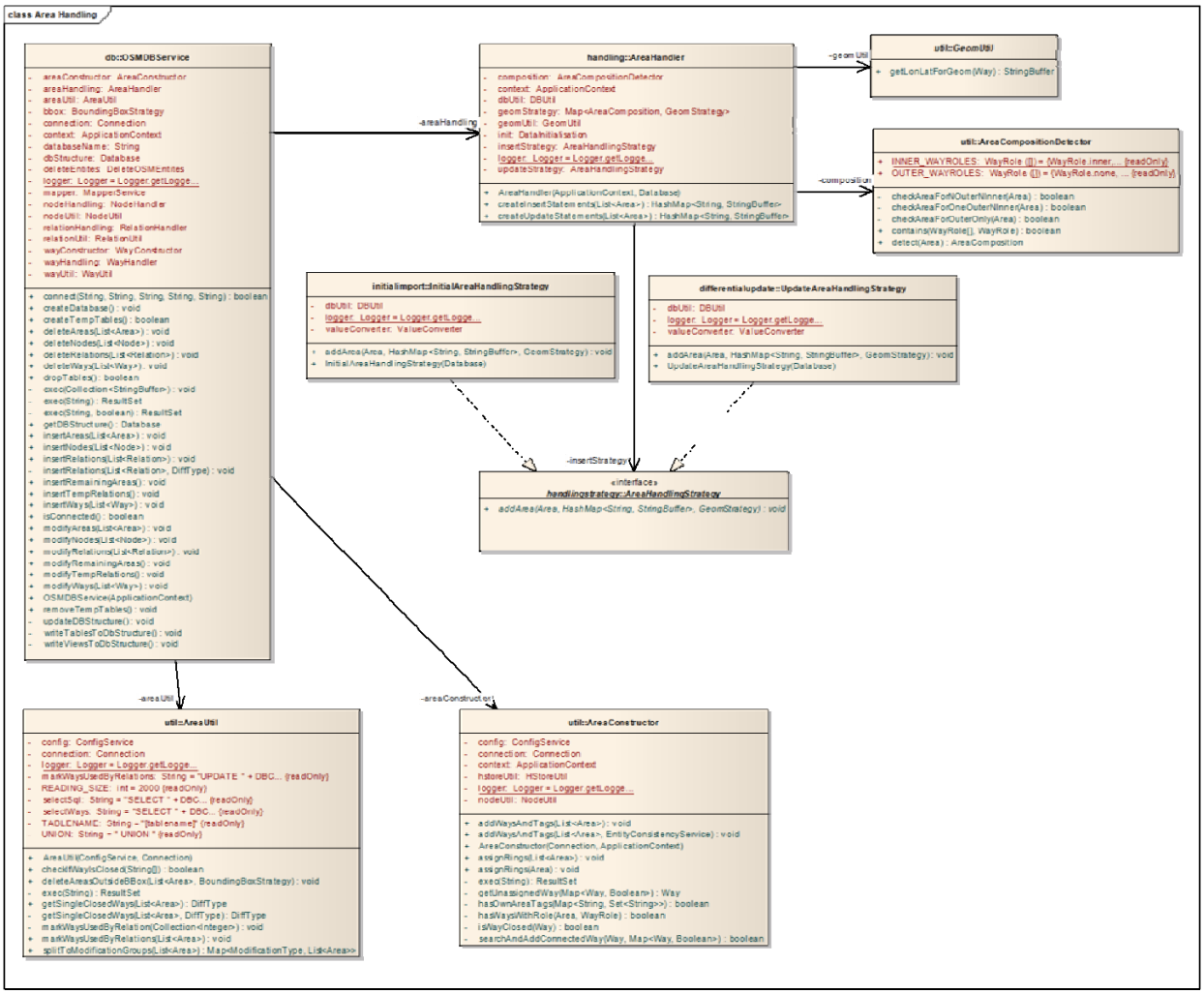


Figure 30: Implementation Area Handling

3.2.3.11.6.1 SEQUENCE DIAGRAM

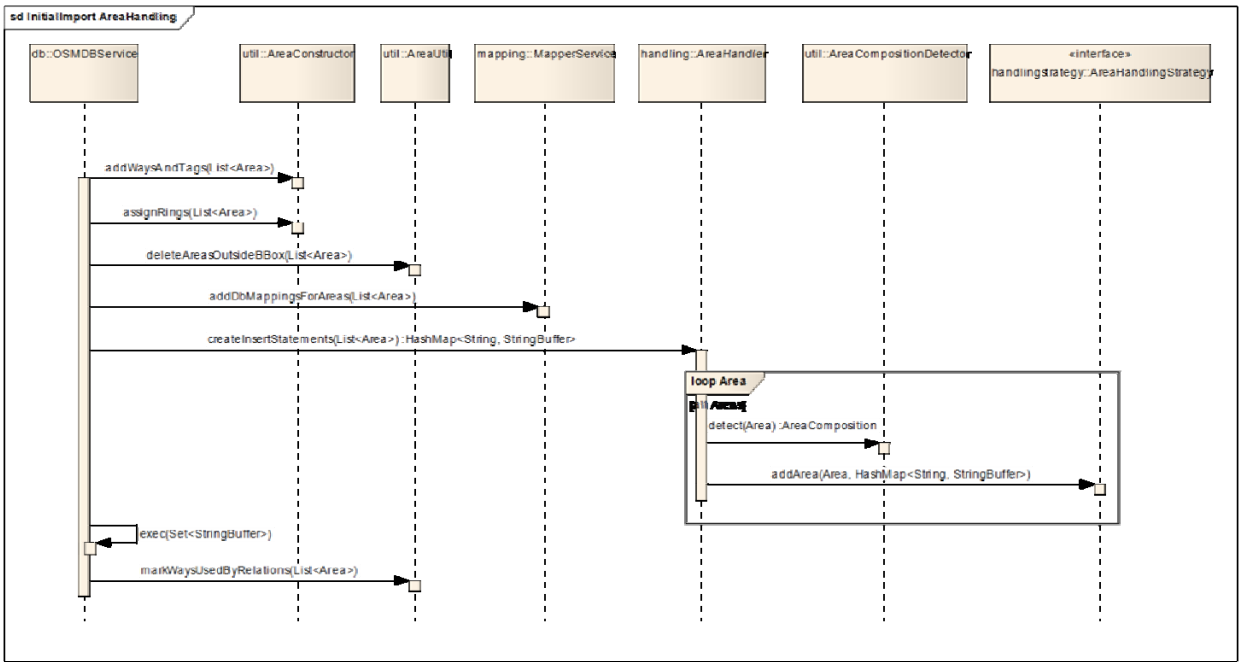


Figure 31: SD Implementation Area Handling

Area handling is a much more complex job than node or way handling. To create an area object, data must be gathered from different places. The AreaConstructor class helps doing this job by analyzing the current state of an area object and adding more data when needed. The AreaCompositionDetector is used just before creating the final sql statement. Depending on the composition of an area, the value for the geometry column in the database looks different.

3.2.3.11.6.2 FLOWCHART INITIAL IMPORT / DIFFERENTIAL UPDATE

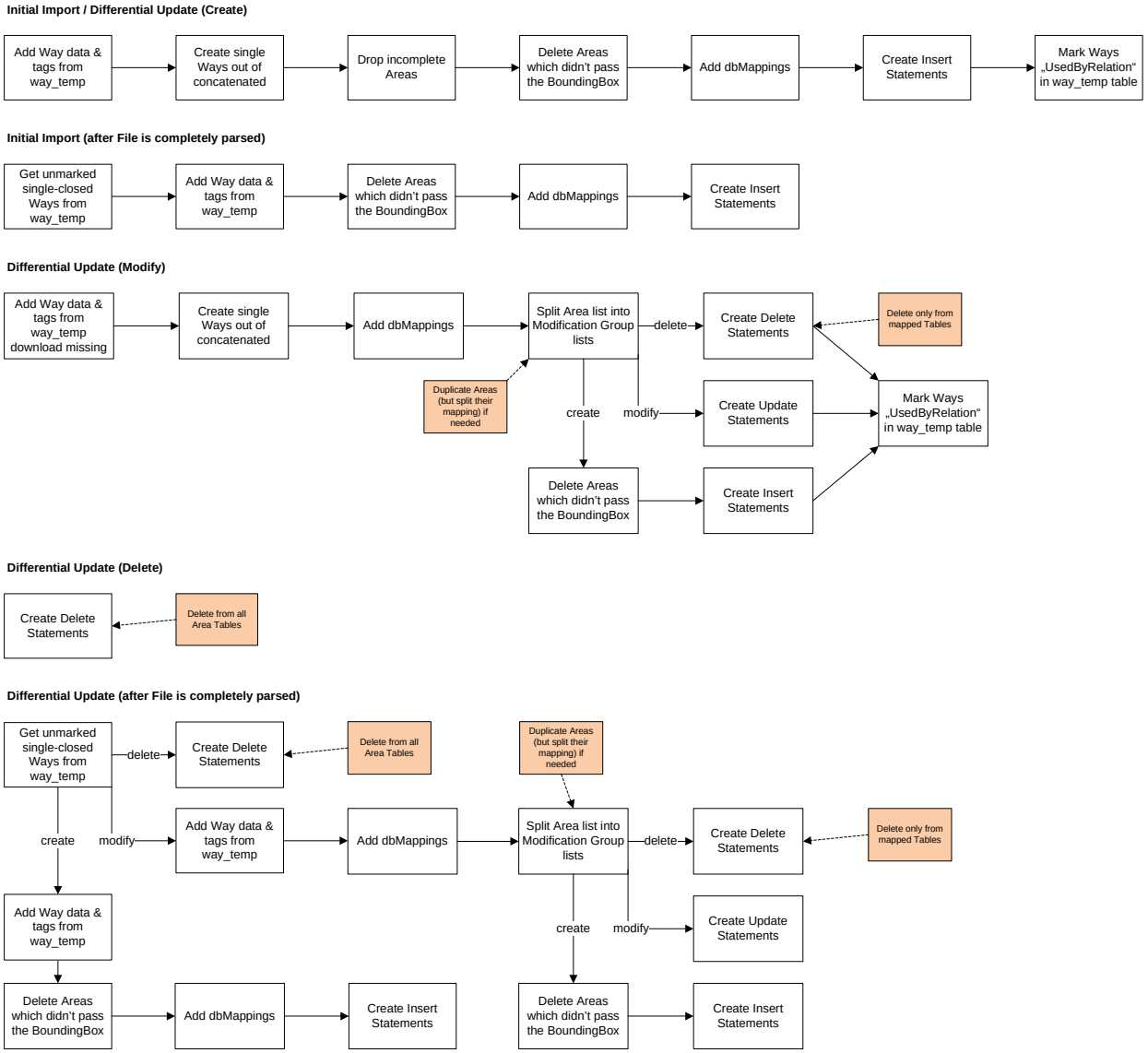


Figure 32: Flowchart for area handling on initial import / differential update

The basic information flow does not differ from area handling and the others. A specialty is that during an import process only areas from OSM relations are created. Once the whole file is finished parsed, the way_temp table is looked consulted to find any single closed ways which are not used by a relation (that's why they got marked during the import process). Many areas come from single closed ways like small lakes, forests etc.

3.2.3.11.7 AREA CONSTRUCTION

Here is a brief flowchart of how an area object is constructed from an OSM relation entity:

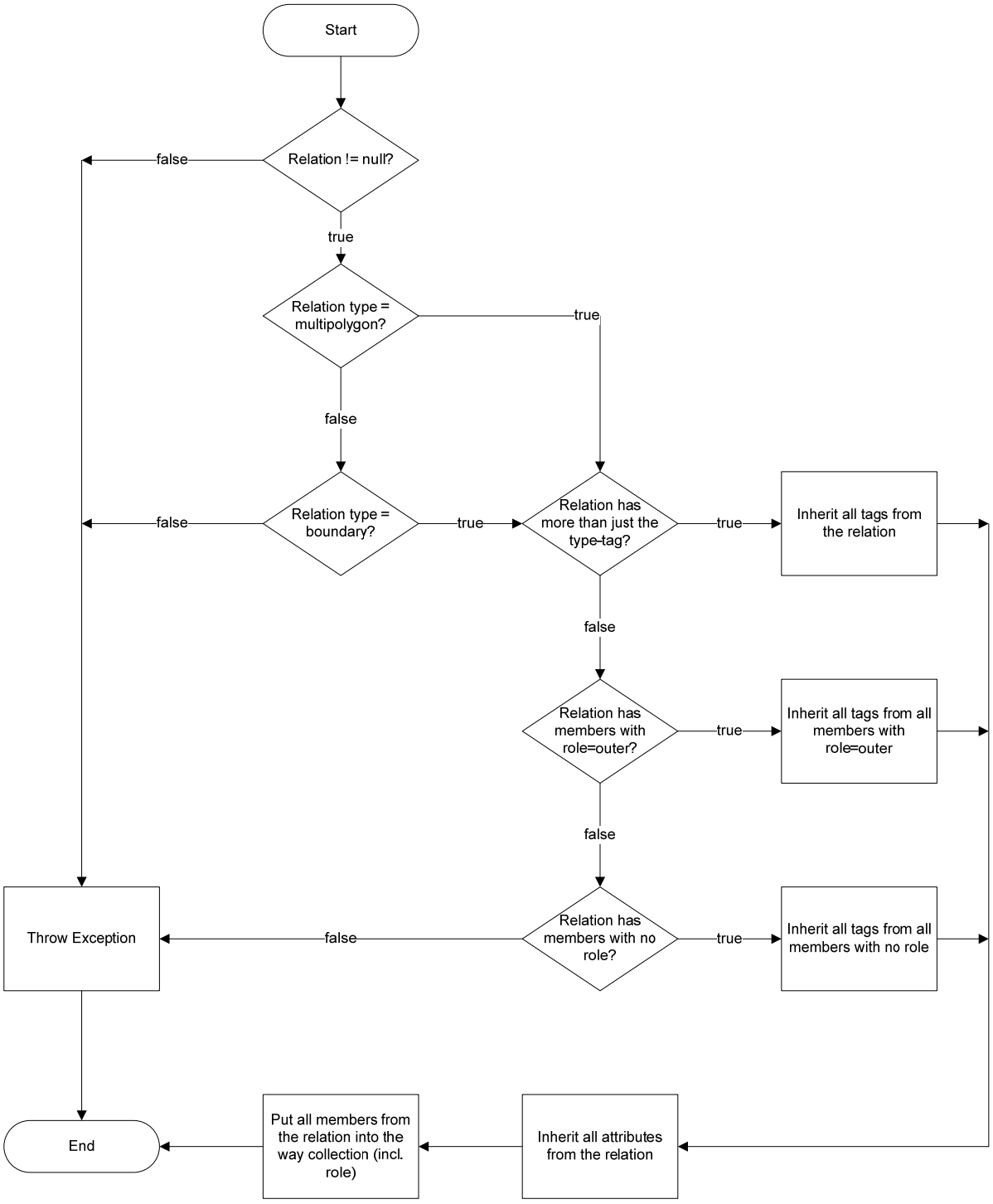


Figure 33: Area construction from relation

3.2.3.12 RELATION HANDLING

3.2.3.12.1 FUNCTIONALITY

Here a list of the functionality implemented on relation handling:

- Locate already imported referenced members in multiple tables and create the SQL statements for the join tables.
- Keep or delete relations for which members are missing in the database depending on user choice.

3.2.3.12.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method insertRelation(..) and modifyRelation(..)
ch.hsr.osminabox.db.handlingstrategy	Relation handling strategy
ch.hsr.osminabox.db.initialimport	Creation of the initial relation values / join entries
ch.hsr.osminabox.db.differentialupdate	Creation of the update relation values
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.util	Util package, used for retrieving member ids from the database etc.

3.2.3.12.3 IMPLEMENTATION RELATION HANDLING

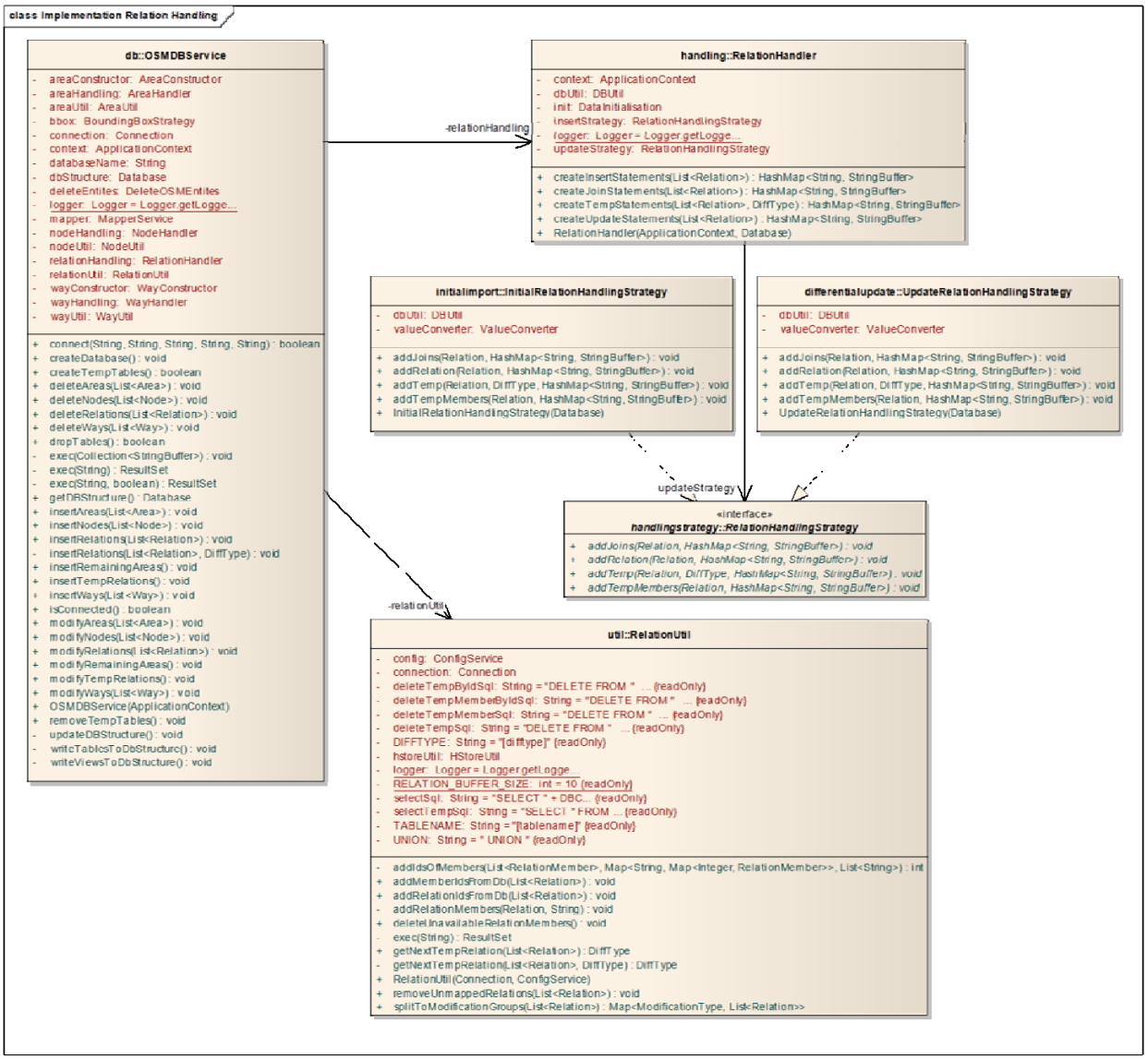


Figure 34: Implementation Relation Handling

3.2.3.12.3.1 SEQUENCE DIAGRAM

Relations are inserted directly into the relation_temp table and their members into the relation_member_temp table. This is because for creating the join entries for the reference tables all members must be inserted in the databes already. Since relations can reference other relations, the ones that doesn't must be inserted before the others so their primary key can be retrieved when creating the join table entry. The following sequence diagram shows the "import" process after all relations are inserted in the relation_temp table.

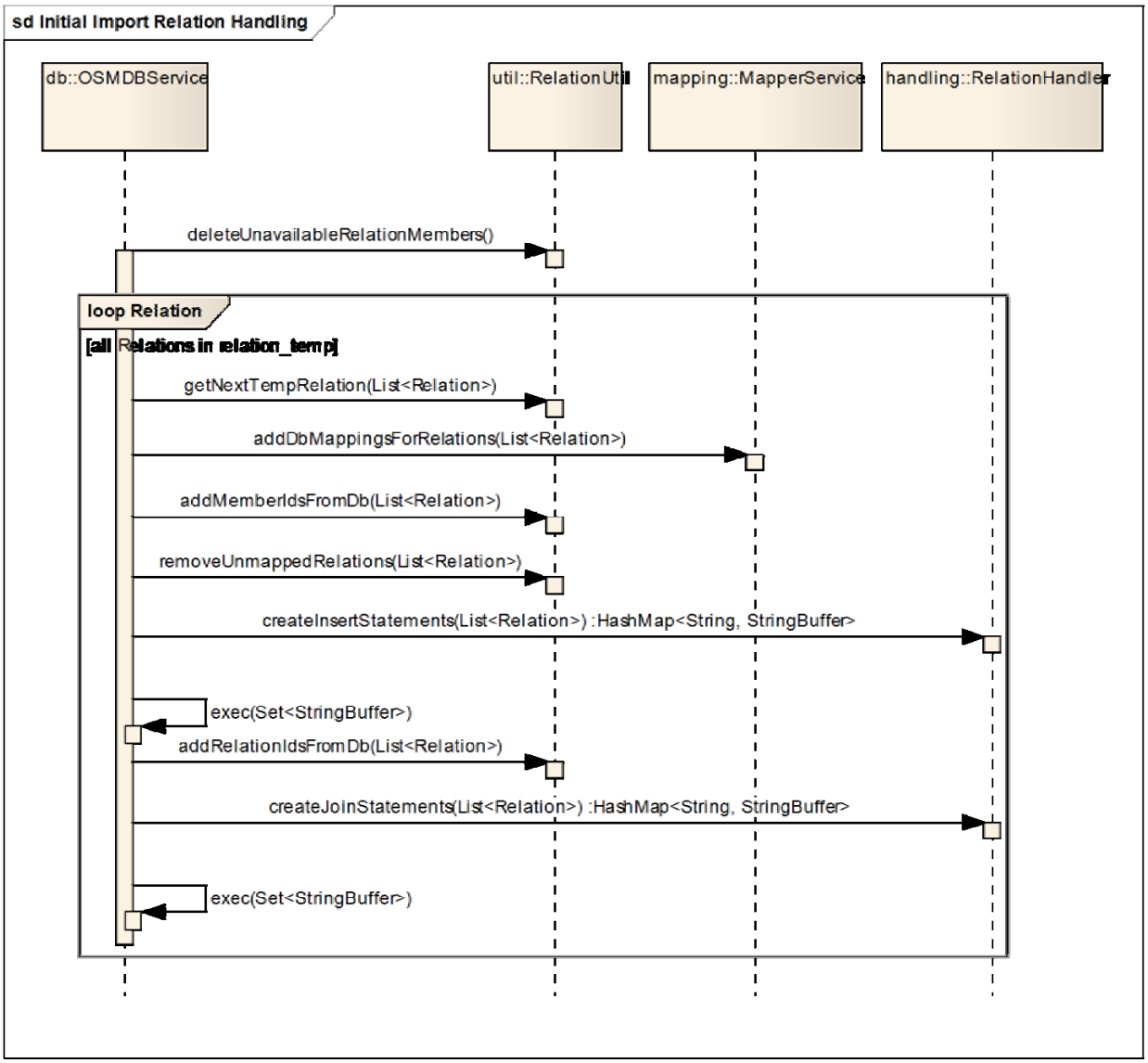


Figure 35: Implementation Rrea Handling

Important is the getNextTempRelation method in the RelationUtil class. It returns a List of Relations which don't reference any unprocessed relations. This assures that no member data, which is available in the database, is missing on a relation.

3.2.3.12.3.2 FLOWCHART INITIAL IMPORT / DIFFERENTIAL UPDATE

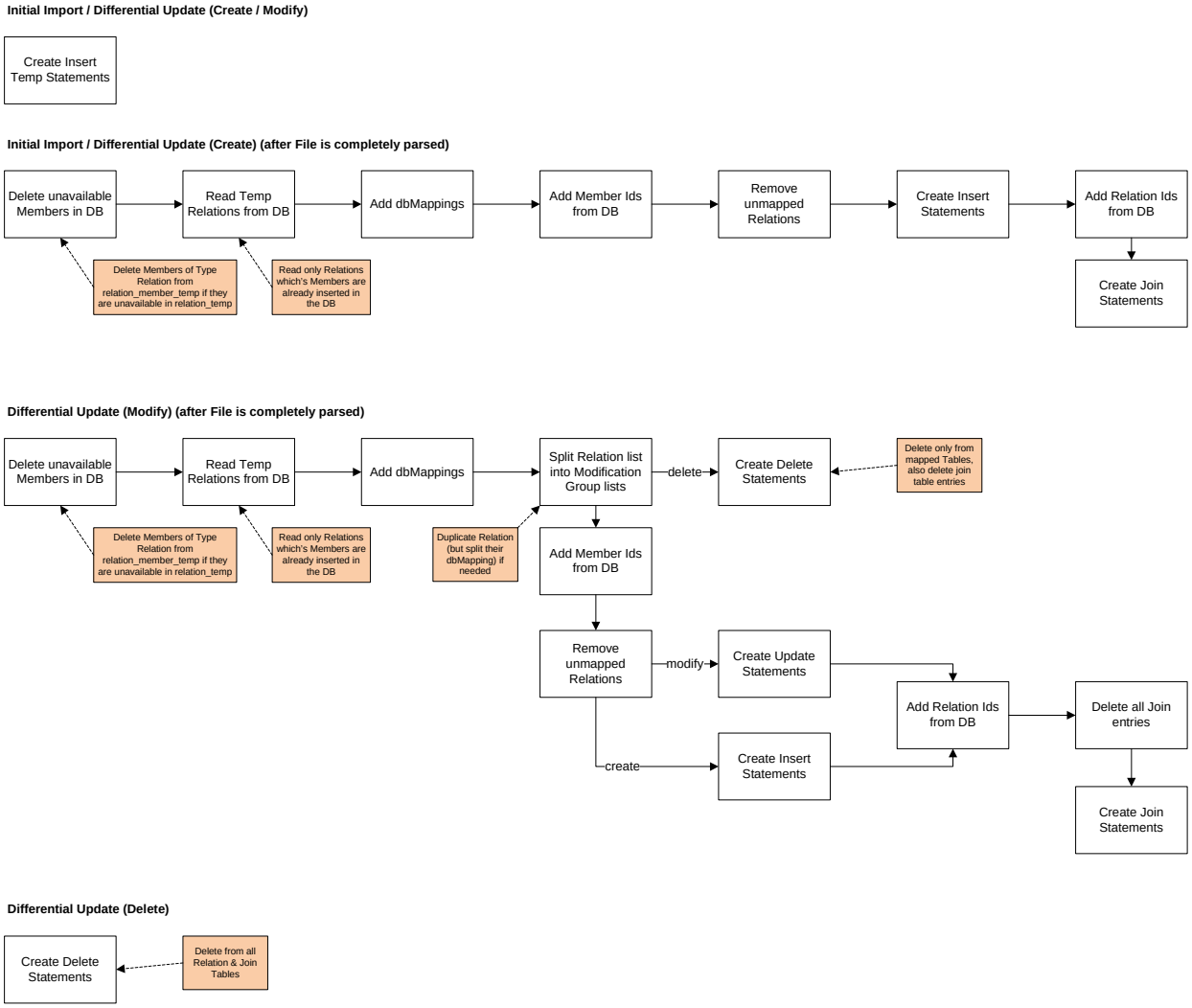


Figure 36: Flowchart for relation handling on initial import / differential update

Processing Relation is a lot different than inserting any other OSM datatype. Aside from displacing the import of relations to the very end of the import procedure, the following changes are important:

- Before processing any relations, all invalid ones (which have no members or invalid references) must be deleted.
- After the dbMappings are added, all related tables from a relation mapping are looked up and referenced members of the relation are searched by their osmId. The found primary keys are taken for the later creation of join entries.
- If a relation needs all its members to be found in the database but this is not the case, the relation is dropped from the import process. More specific: that dbMapping is deleted but since most relations are only mapped to one dbMapping, this leads to the same.
- The remaining relations are inserted in their mapped table and the created primary keys are taken again.
- Now all relevant data is gathered to create the join entries.

In a differential update process when a relation must be modified, all its join entries are deleted and recreated instead of updated due to simpler implementation.

3.2.3.13 DELETE HANDLING

Within a differential update file aside from creating new or modifying existing entities, the third modification can be the deletion of them. According to OSM if an entity is to be deleted, no additional information about the entity needs to be provided in the update file than the osm id. Since such a deletion is equal for all entities, a single DeleteOsmEntity class can do the job.

3.2.3.13.1 IMPLEMENTATION DELETE HANDLING

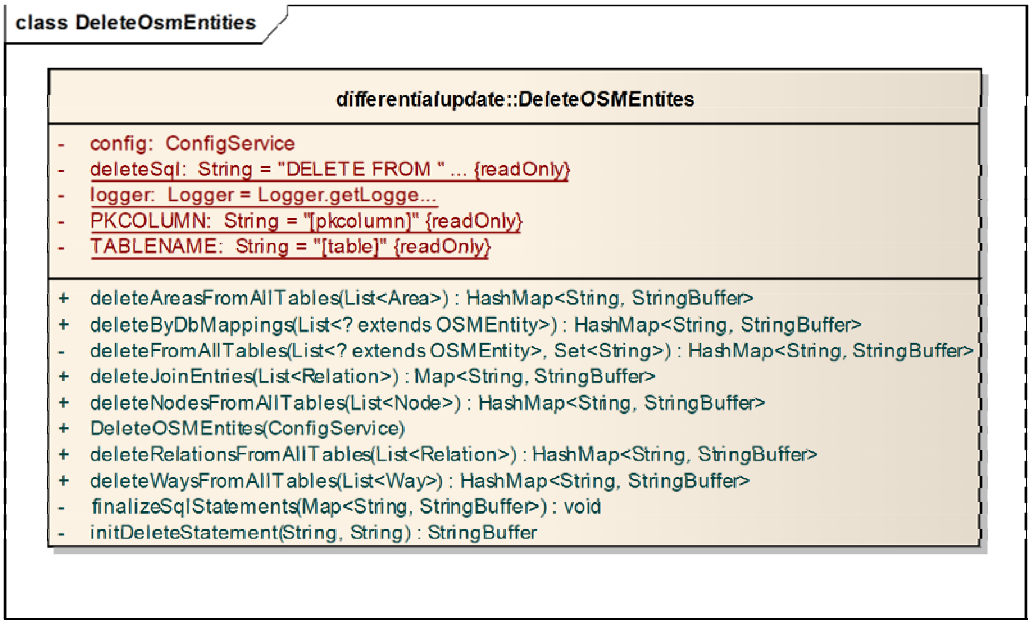


Figure 37: Delete Handling of Osm Entites

If an OSM entity must be deleted, it is deleted from every table that matches the entities type. If an entity needs to be deleted because it is modified and no longer belongs to a table it was mapped to before, that entity must only be deleted from that specific, mapped table. Both deletions are done in the DeleteOsmEntites class.

3.2.3.14 ENTITY CONSISTENCY SERVICE

3.2.3.14.1 FUNCTIONALITY

The EntityConsistencyService is needed because on an update process it is likely that some references are missing in the update file. Therefore, they need to be downloaded via the OSM api.

3.2.3.14.2 INVOLVED PACKAGES

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

3.2.3.14.3 IMPLEMENTATION

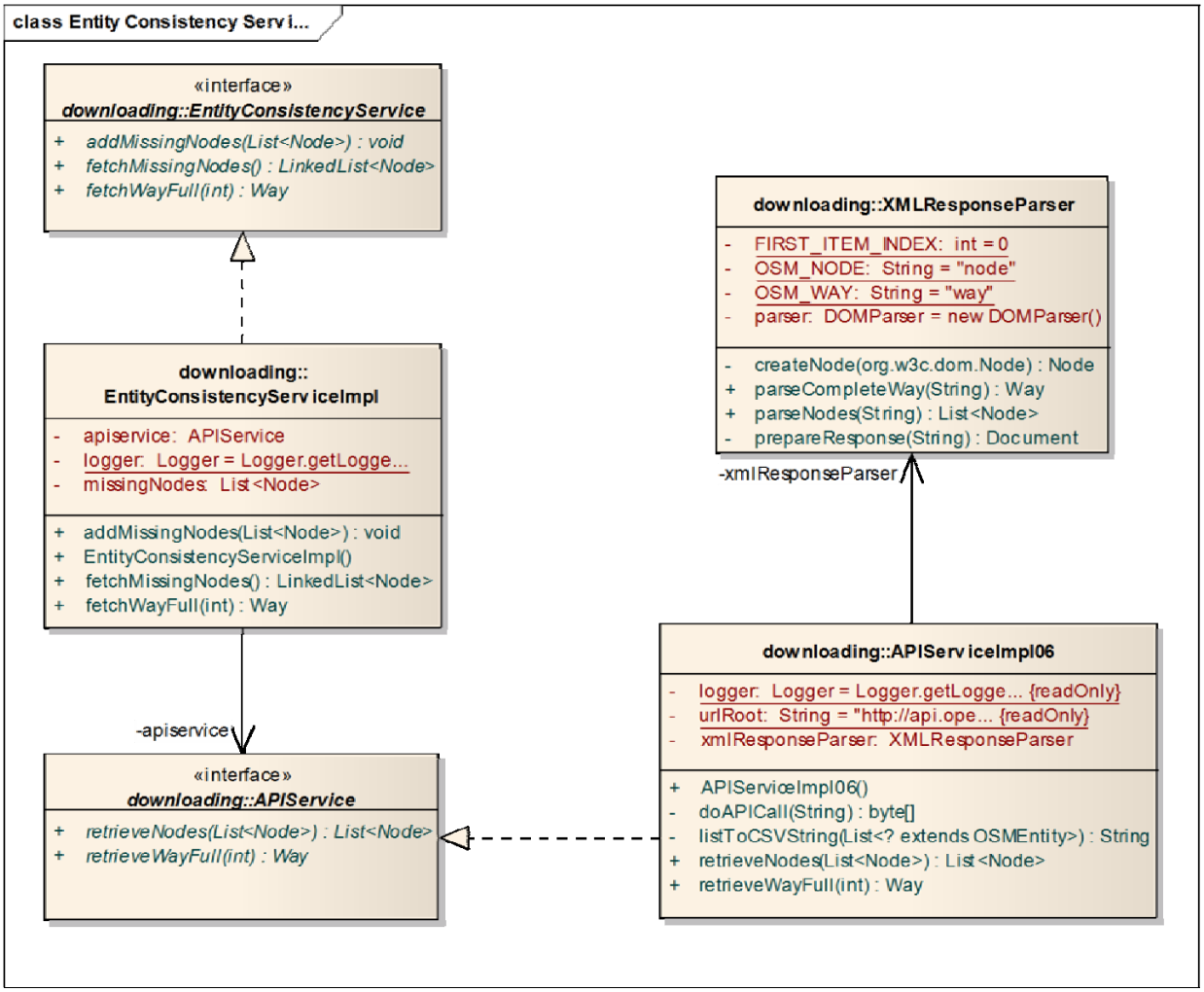


Figure 38: Class diagramm entity consistency service

The EntityConsistencyService checks the integrity of a way, area 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.

3.2.3.15 DB-SCHEMA AND MAPPING HANDLING

Figure Figure 39: Overview about the Schema Mapping configuration shows the physical architecture of the way to configure OpenStreetMap-in-a-Box. For more information about configuring Schema Mapping File refer to 3.5 Usermanual.

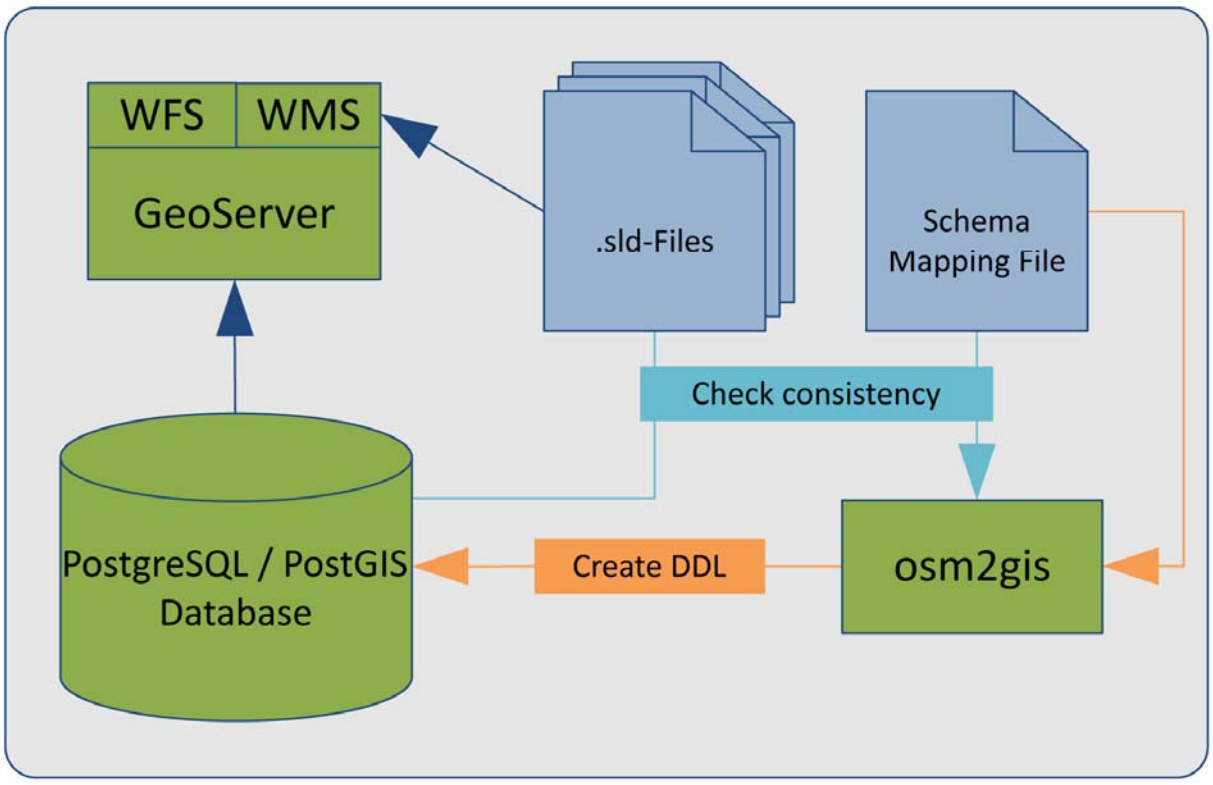


Figure 39: Overview about the Schema Mapping configuration

3.2.3.15.1 FUNCTIONALITY

The complete handling of db-schema and mapping configuration is done by the schemamapping package.

It provides a service to generate a SQL from the Schema Mapping File. This functionality is located in the package xml2ddl.

The consistency package has functionality to check Schema Mapping File, Database sctructure and GeoServer SLD files against each other.

3.2.3.15.2 INVOLVED PACKAGES

The schemamapping package provides three main functionalities where can be found in the following 3 packages.

- schemamapping
  - consistency
  - xml2ddl
  - ConfigService

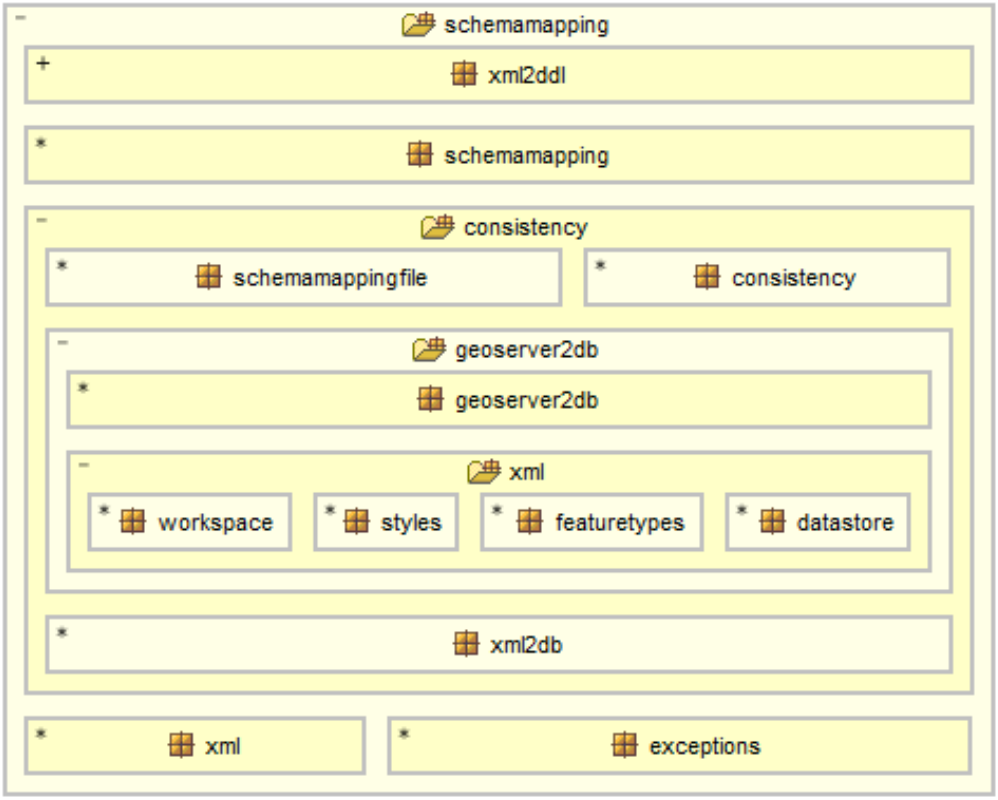


Figure 40: Package structure

3.2.3.15.3 INTRODUCING CONFIGSERVICE

In the package schemamapping exist a ConfigServiceImpl class. This class is part of the ApplicationContext and gives possibilities to read important information from the Mapping Configuration File.

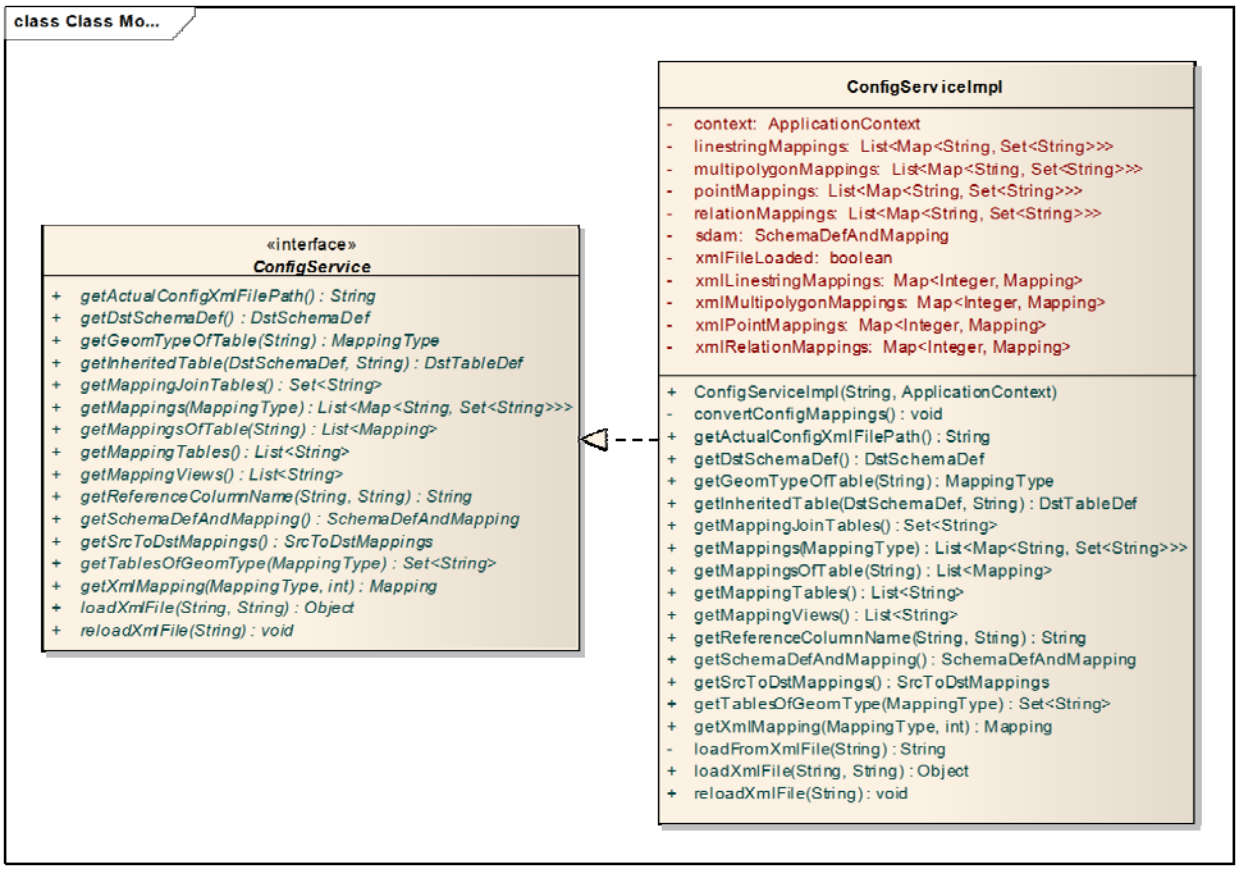


Figure 41: ConfigService

3.2.3.15.4 INTRODUCING CONSISTENCYSERVICE

In the Package schemamapping exists a ConsistencyService. This class is part of the ApplicationContext and gives possibilities to start ConsistencyChecks.

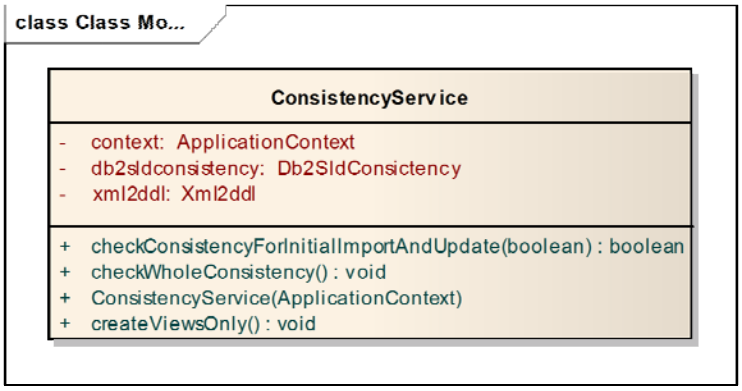


Figure 42: ConsistencyService

3.2.3.15.5 CONSISTENCY CHECKS

There are four consistency checks:

1. (Errors) src_to_dst_mappings to dst_schema_def in mappingconfiguration
    a. This checks consistency inside the Schema Mapping File.
2. (Differences) dst_schema_def in mappingconfiguration to DB
    a. This check checks if every table with his columns exists in the DB in use.
3. (Errors) GeoServer feature type (inc. SLD) to DB
    a. This includes the feature types and the SLD files from the GeoServer data folder.
4. (Hints) mapping configurations to GeoServer (*.SLD)
    a. This is a check from the Mappings to the feature type and the SLD files. There are only hints in this check, because it is not needed to configure SLD files for every mapping.

There is a possibility to start this 4 Consistency check with osm2gis --consistency.

3.2.3.15.5.1 MAPPING SCHEMA FILE TO DB

The following flowchart illustrates the logic of the consistency check:

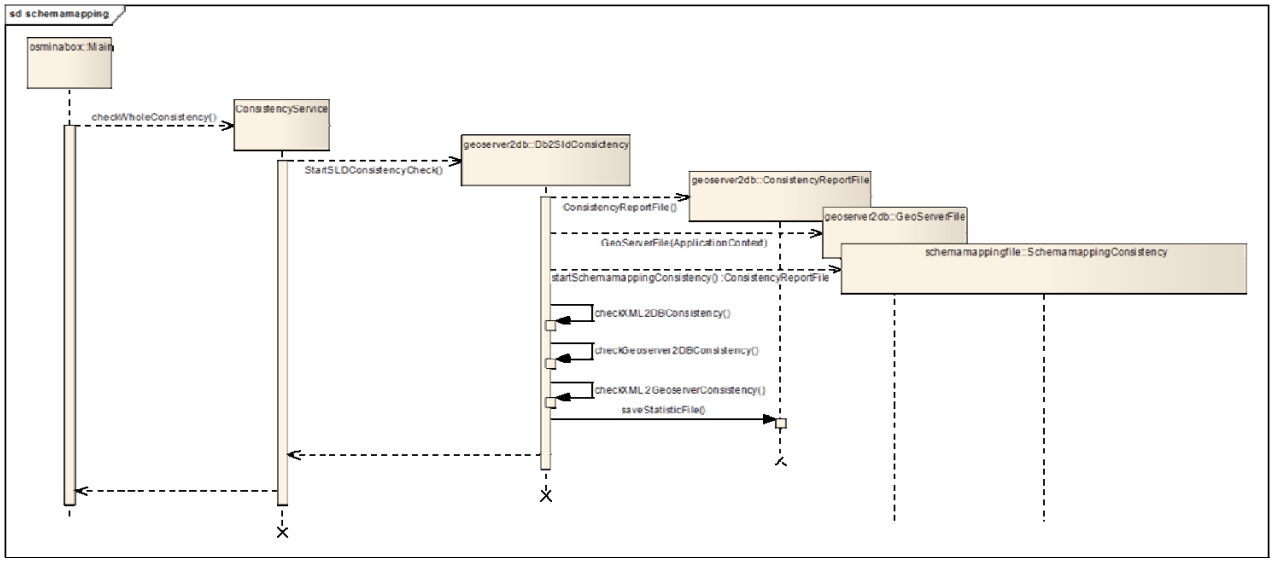
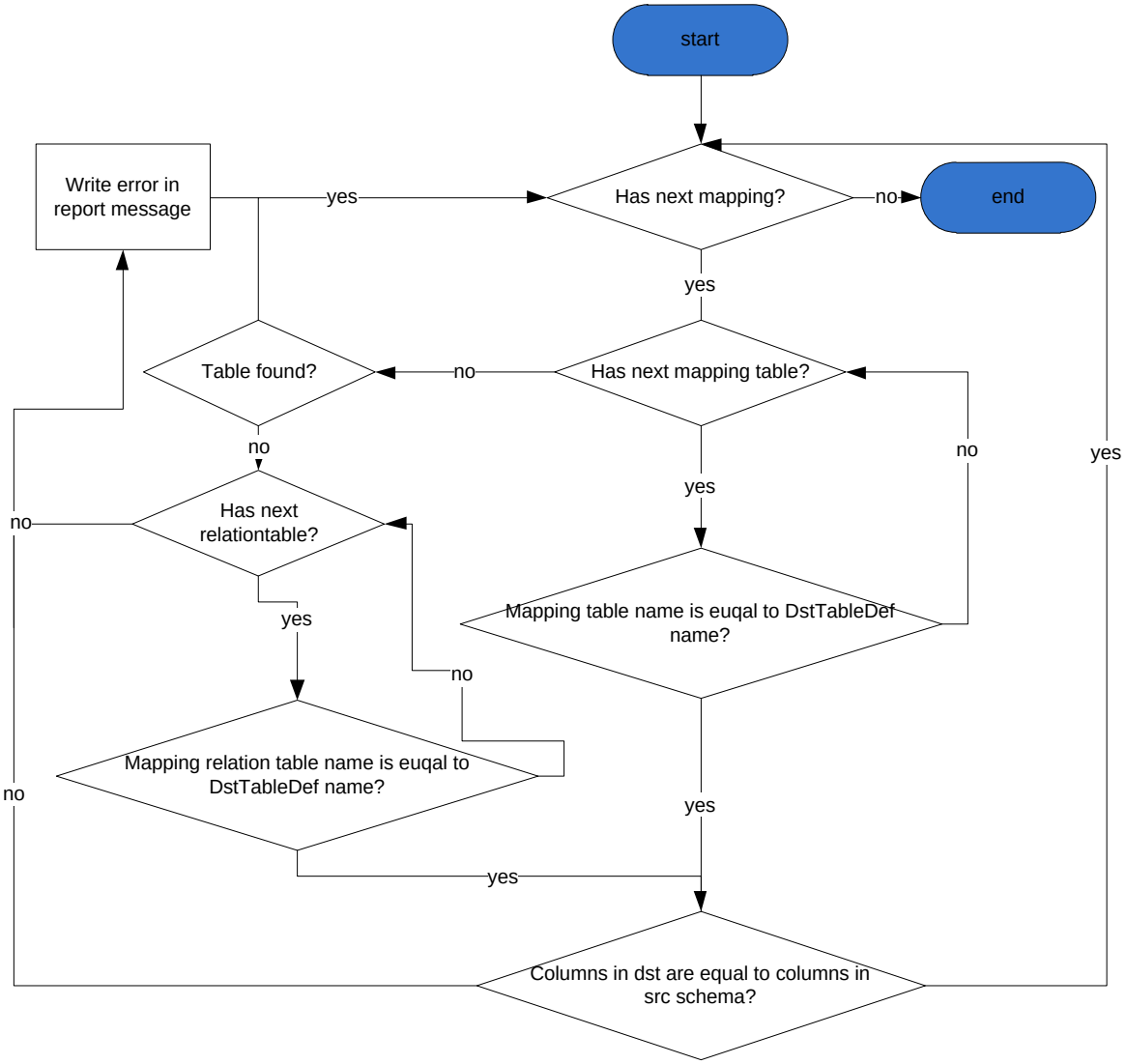


Figure 43: Sequence of consistency check

3.2.3.15.5.2 SRC_TO_DST_MAPPINGS TO DST_SCHEMA_DEF IN MAPPINGCONFIGURATION



3.2.3.15.5.3 DST_SCHEMA_DEF IN MAPPINGCONFIGURATION TO DB

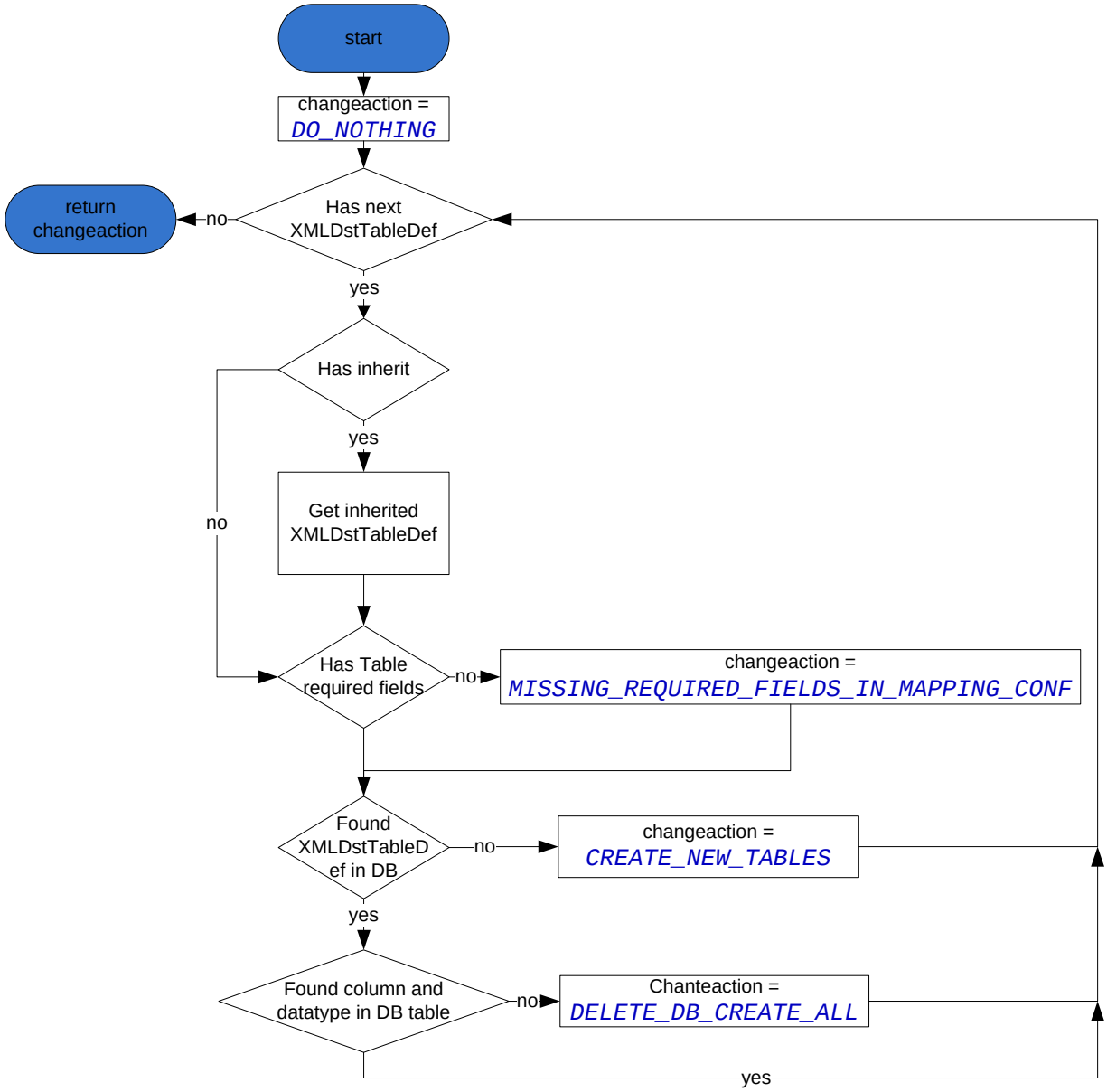


Figure 44: Flowchart mapping schema file to db

3.2.3.15.5.4 GEOSERVER FEATURE TYPE (INC. SLD) TO DB

The following flowchart illustrates the logic of the GeoServer to DB consistency check:

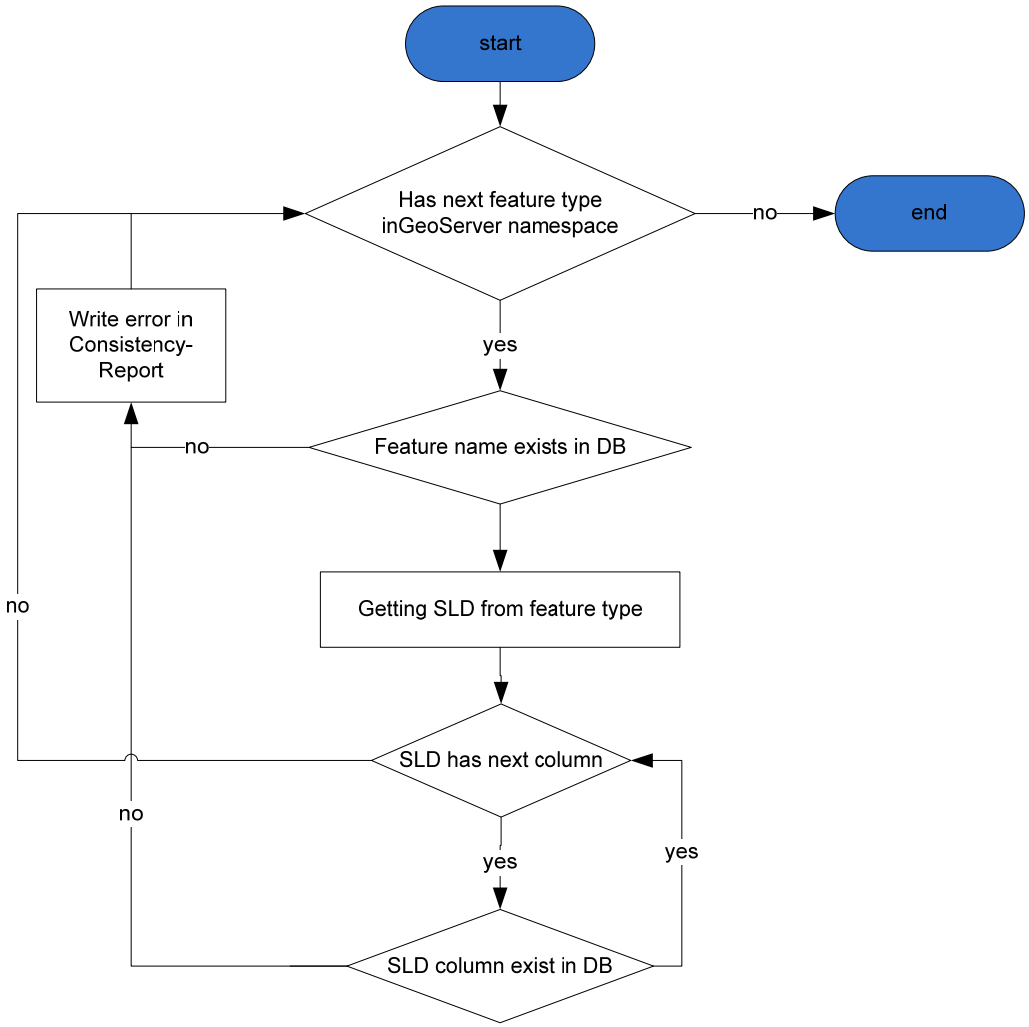


Figure 45: Flowchart GeoServer to DB

3.2.3.15.5.5 MAPPING CONFIGURATIONS TO GEOSERVER (*.SLD)

The following flowchart illustrates the logic of the Mapping Schema File to GeoServer consistency check:

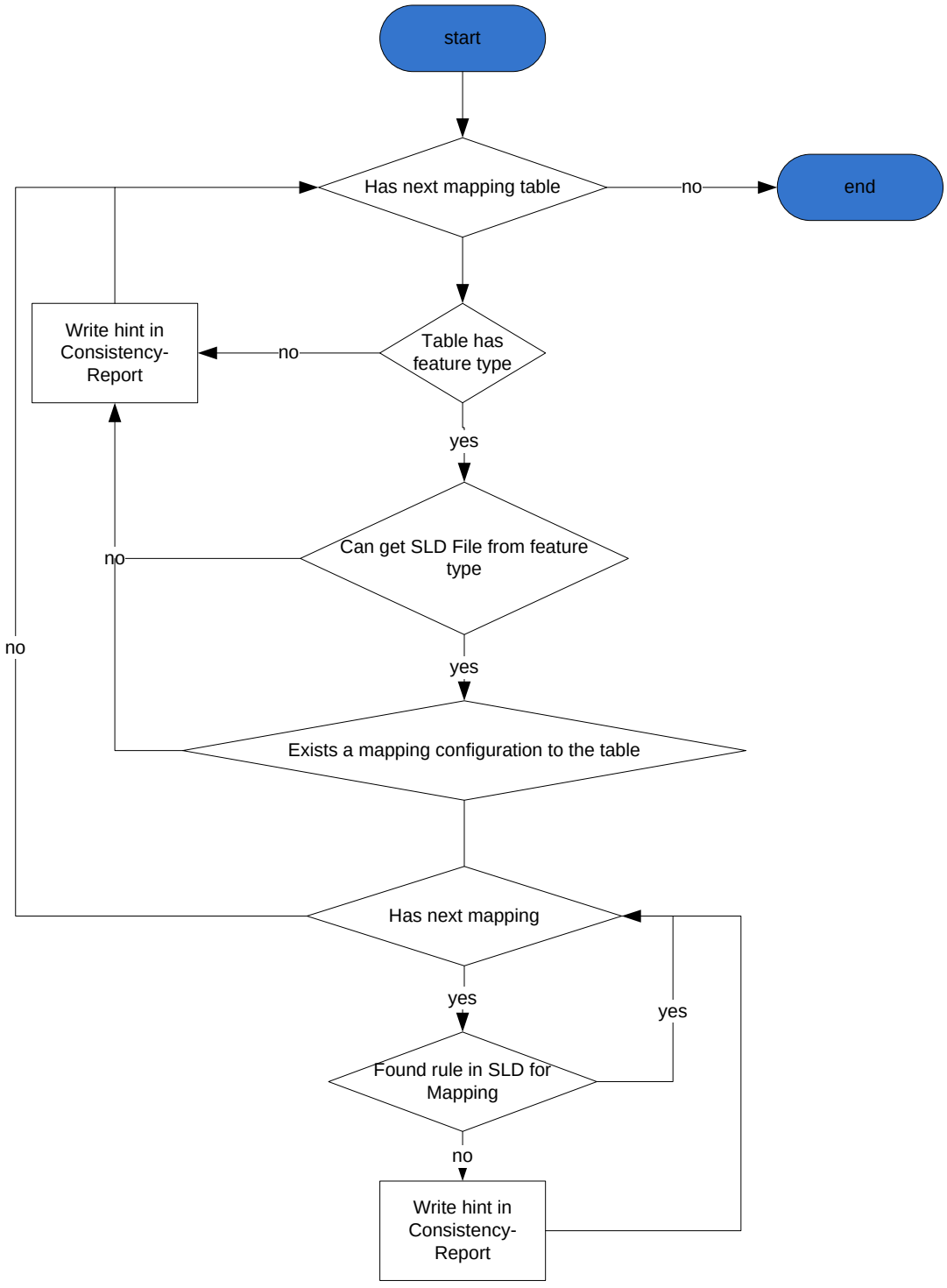


Figure 46: Flowchart mapping schema file to GeoServer

3.2.3.15.6 MAPPING SCHEMA FILE TO DDL

In every initial-import and update or scheduled update the xmldbconsistency checks if the Schema Mapping File is consistent and if there are differences in the Schema Mapping File to the DB structure. If there are differences between Schema Mapping File and DB, it decides what should happen to solve this inconsistency.

How the check of the Schema Mapping File to the DB works is described in the chapter 3.2.3.15.5.1.

This sequence diagram represents the start of an initial-import. The initial-import starts the startGeneration method on xml2ddl in if this method gives back true then the initial-import can start. If it returns false then the application has to shutdown, because there exists an inconsistency!

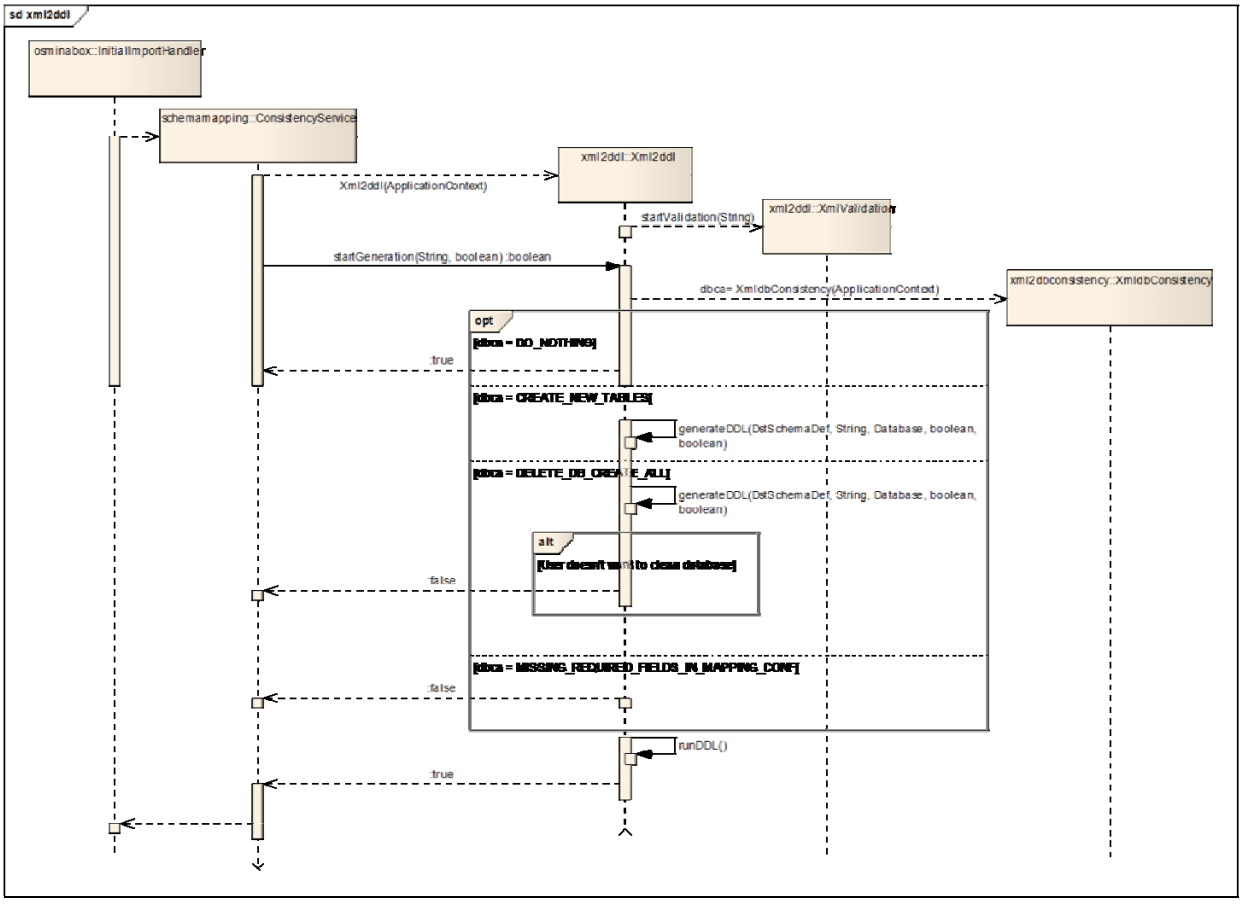


Figure 47: Sequence of start of the initial import consistency check

On an update or scheduled update it is the same sequence. The only difference is that on an update or scheduled update not the initialImportHandler starts consistency check but the UpdateScheduler.

3.2.4 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 you can be found here: http://wiki.openstreetmap.org/wiki/Tags

In the Schema Mapping File there's the <mapping>'s section. A User can define which values from which keys should be inserted in which column of his table. He also can define a constant value for a certain column. This makes sense, if he has several mappings which are inserted into the same table (for example different pois).

All information about how to create those mappings, see the user manual.

If you are unsure about GIS standards, see the whitepaper from Jochen Topf "OpenStreetMap Data in Standard GIS Formats".

3.2.4.1 MAPPING SERVICE

3.2.4.1.1 FUNCTIONALITY

The Mapping Service provides the functionality to map an OSM-Entity to one or many corresponding database table according to its tags and entity type. It uses the mappingconfig.xml file described in the chapters above to look up the mappings.

3.2.4.1.2 INVOLVED PACKAGES

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

3.2.4.1.3 IMPLEMENTATION

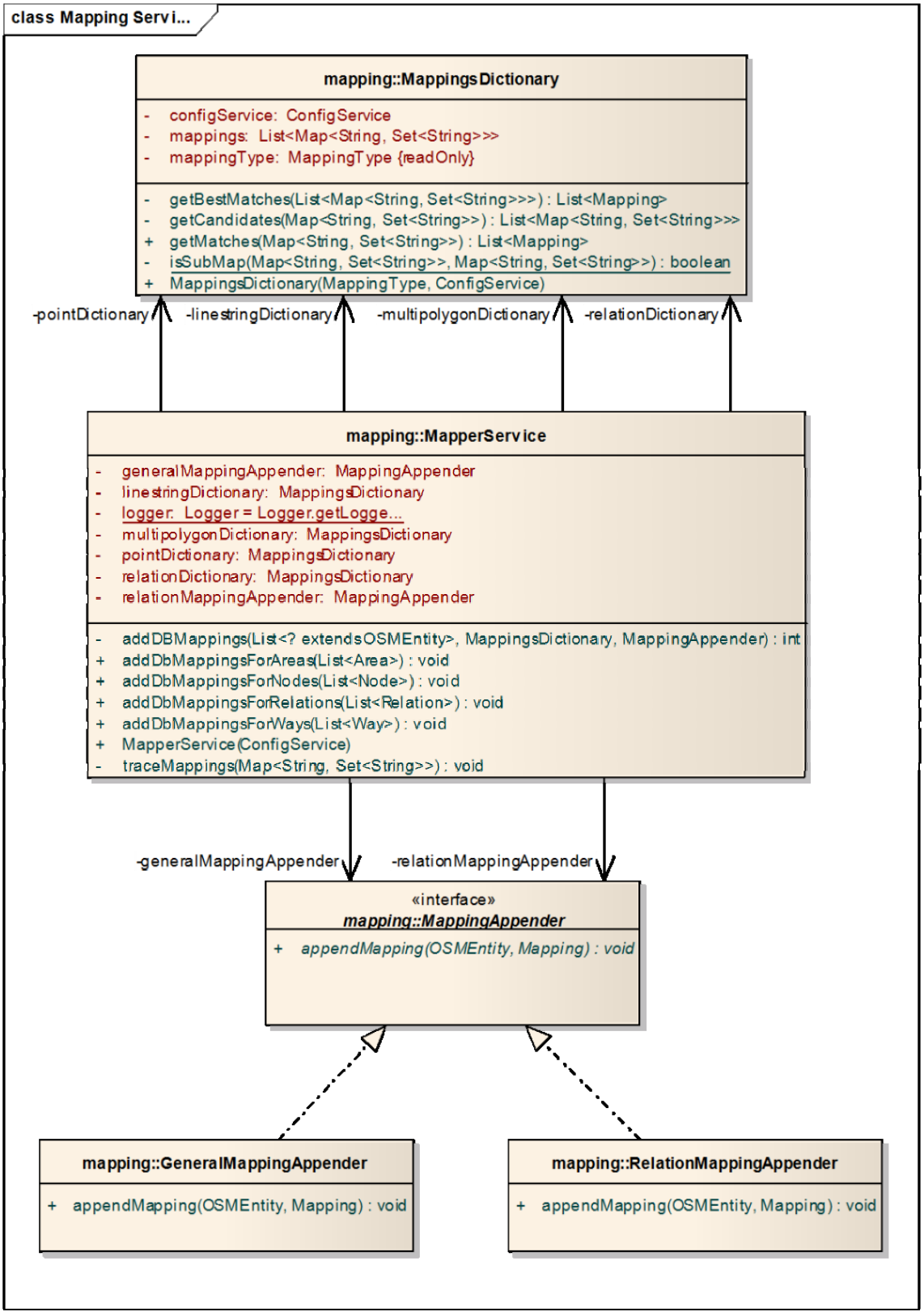


Figure 48: Mapping Service

Explanation:

The MapperServiceImpl class provides the functionality to map an entity to one or multiple tables in the database. It contains a tags dictionary which holds the information about which OSM-Tag combination leads to a specific table. The MapperService identifies all mappings (since an OSM-Node can represent more than one entity on a map) and sets them in the OSMEntity class.

For the reason that relations need a much more complex dbMapping class, a strategy pattern is used for adding the relevant mapping informations.

3.2.5 MAPPING CONFIGURATION

The whole mapping configuration is done by a Schema Mapping File (XML).

For information about handling this file read chapter 0.

3.2.5.1 XML-SCHEMA

The XML-Schema is defined by the mappingconfig.xsd file that is in the config folder. Osm2gis validate this mapping configuration file on every reading.

For more information on how to write this file please read the usermanual chapter 2.

3.2.6 DATABASE

As database software 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'. To store all (unused) tags from entities, an associative Array Data type namely HStore is used. The necessary SQL is executed at the start of an import if HStore doesn't exist yet in the database.

3.2.6.1 DATABASE SCHEMA

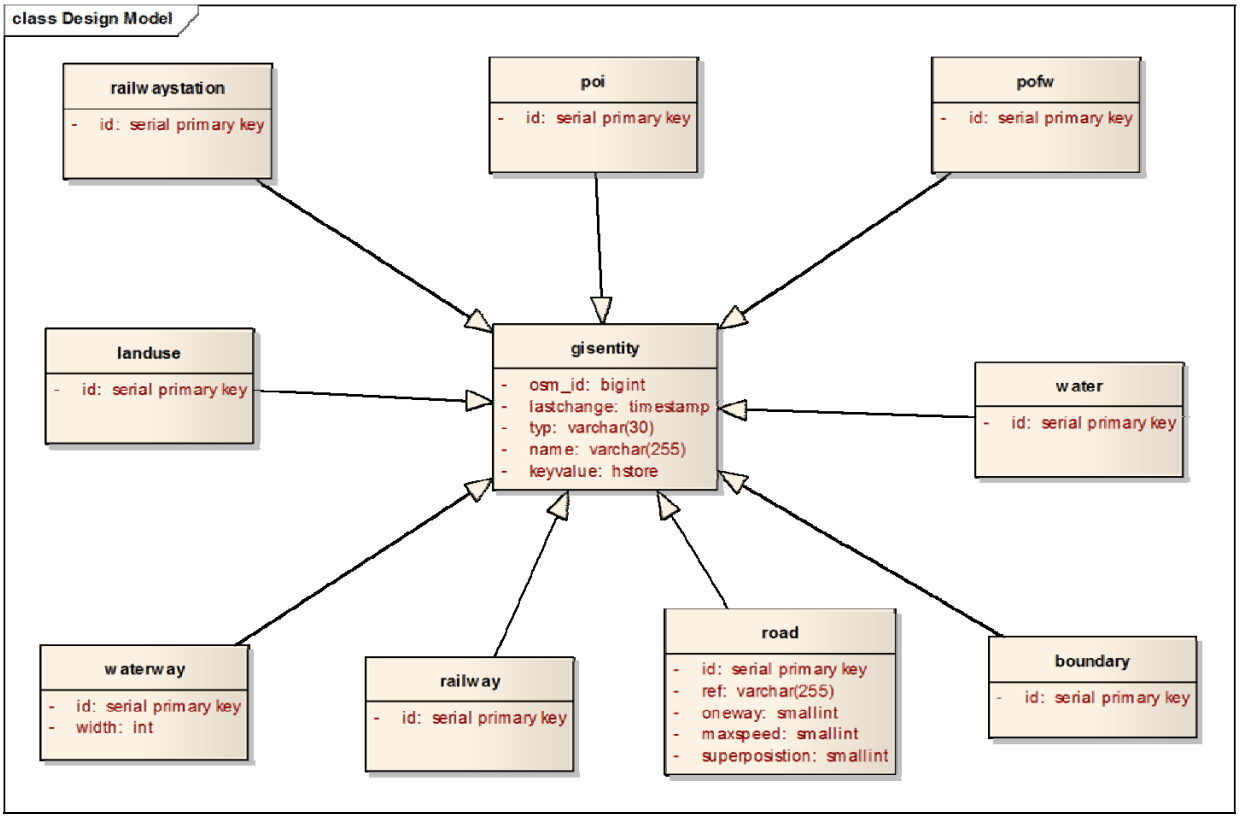


Figure 49: Database schema example

This diagram shows an example of the database schema. It's taken from the standard configuration file distributed with version 2.0.

All entites are a generalization of gisentity. The id has to be set in every entity itself, otherwise the serial id is 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. The type of the inserted data must be defined. It can vary from POINT (Node), LINESTRING (WAY) and MULTIPOLYGON (Relation / closed Ways).

3.2.6.2 DATABASE SOFTWARE

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

Software	Version	Link
PostgreSQL	8.4	http://www.postgresql.org/
PostGis	1.5	http://postgis.refractions.net/

3.3 PROJEKTMANAGEMENT

3.3.1 PROJEKTORGANISATION

Das Projektteam für die Bachelorarbeit besteht aus Meier Andreas und Zimmermann Joram. Das Projekt erlaubt es, die Verantwortlichkeiten grob auf die zwei 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.

3.3.1.1 ORGANISATIONSSTRUKTUR

Name	Email	Verantwortlichkeit
Meier Andreas	ameier@hsr.ch	Konsistenzprüfung, Geoserver, Installer, DDL Generator, Config-File
Zimmermann Joram	jzimmerm@hsr.ch	Initial Import Relation Handling und Differential Update

3.3.1.2 BETREUUNG

Name	Email	Funktion
Stefan Keller	sfkeller@hsr.ch	Betreuung
Hans Rudin		Gegenleser
Claude Eisenhut		Externer Gegenleser

3.3.2 MANAGEMENT ABLÄUFE

3.3.2.1 ZEITMANAGEMENT

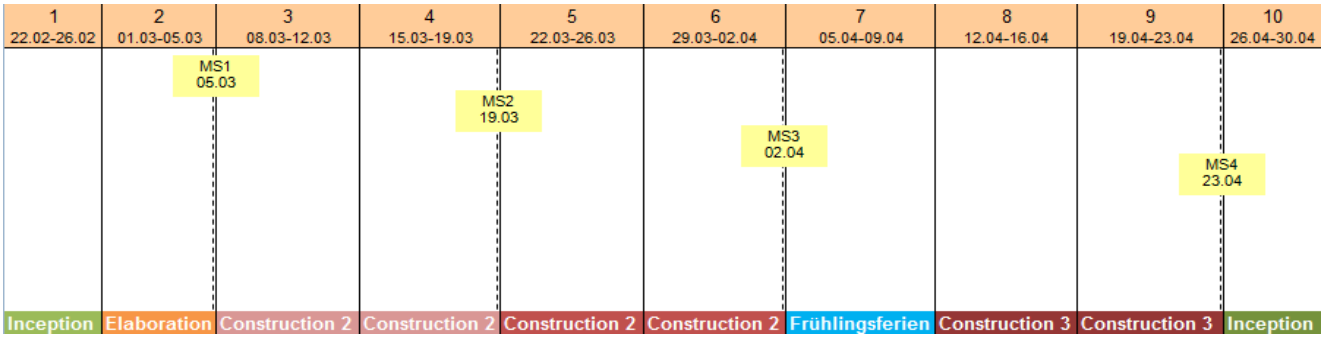
Für die gesamte Bachelorarbeit stehen 17 Wochen zu Verfügung, die 7 Woche(Frühlingsferien) wird nicht verplant jedoch als Puffer verwendet, Woche 16 werden Herr Meier und Herr Zimmermann den CCNA Kurs besuchen daher stehen 15 Wochen zur Verfügung. Der Projektplan wurde so ausgelegt, dass die 15 Wochen vollständig ausgenutzt werden. Die als Reserve eingeplante Zeit in Woche 14 und 15 des Projektes wird entweder als Puffer für allfällige Verzögerungen, oder für die Implementation weiterer Features genutzt.

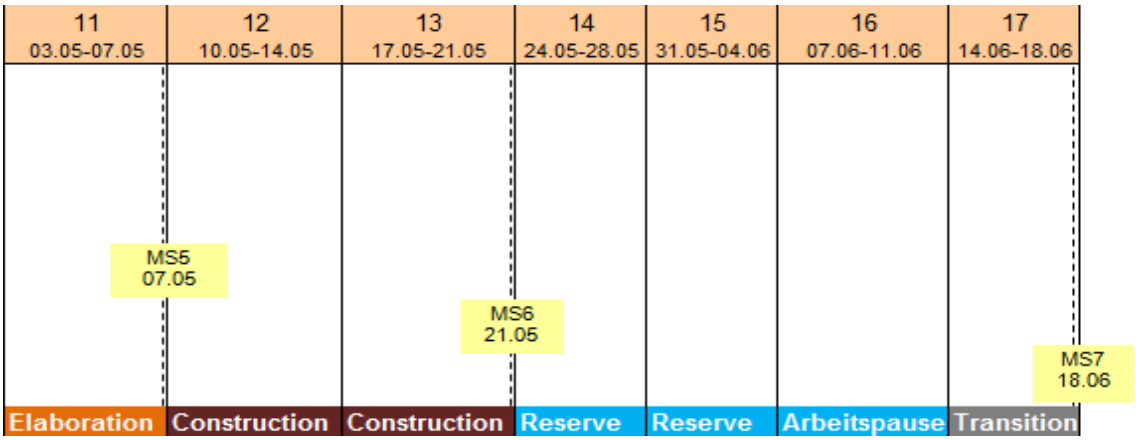
3.3.2.2 PROJEKTPLAN

3.3.2.2.1 ZEITPLAN

Siehe Projektplan.xlsx für mehr Details.

3.3.2.2.2 ITERATIONSPLANUNG





3.3.2.2.3 MEILENSTEINE

Meilenstein 1:	05.03.2010
• Projektplan in einer ersten Version o Arbeitspakete und Planung Server • Requirements • Risikomanagement • Aufgabenstellung	
Artefakte: - Projektplan - Zeitplan - Risikomanagement - Anforderungsspezifikationen	

Meilenstein 2:	19.03.2010
• Config um Relationhandling erweitern • Relationhandling für initial-import • Erweiterung um Relationhandling in der Konsistenzprüfung und DDL generierung	
Artefakte: - SAD	

Meilenstein 3:	02.04.2010
• Differential Update • Konsistenzprüfung Geoserver	
Artefakte: - SAD	

Meilenstein 4: Server Abschluss	23.04.2010
• GeoServer Maskierungslayer • GeoServer Tiling / Caching • GeoServer Render / Out of Memory Fehler • Funktionstests Win / Unix	
Artefakte: - SAD - Usermanual - Testdokument	

Meilenstein 5:	07.05.2010
• Projektplan erweitern o Arbeitspakete und Planung Mobile WMS Client • Requirements erweitern • Evaluation Mobile WMS Client Library • Prototyp	
Artefakte: - Projektplan - Zeitplan - Anforderungsspezifikationen	

Meilenstein 6: Geoserver überarbeitet	21.05.2010
• Neue SLD für GeoServer wurden erstellt. • Suchfunktion auf die neuen Relations ist im Showcase gegeben. • Diff-Update ist in Zwischenphase	
Artefakte: - SAD - Usermanual - Gesamtdokumentation	

Meilenstein 7: Integration des Gesamtsystems	18.06.2010
• Fertigstellung der Dokumentation • Showcase fertigstellen • Diff-Update fertiggestellt • Version 1.0 RELEASE!	
Artefakte: - Gesamtdokumentation	

3.3.3 ARBEITSPAKETE

3.3.3.1 PROJEKTMANAGEMENT

5.1.1 Projektplan Excel	10 Std.
Beschreibung	Das Excel für den Projektplan erstellen und für die Verwendung vorbereiten. Später wird immer für die nächste Iteration vorgeplant. Übertragen der Zeiten aus den persönlichen Zeiterfassungen in den Projektplan.xlsx wird jeweils am Ende der Woche durchgeführt.
Verantwortlich	jzimmerm
Abhängigkeiten	Nachführen der Zeiten ist abhängig von -> 5.1.3 Excel Zeiterfassung

5.1.2 Projektplan Dokument	12 Std.
Beschreibung	Erstellen des Word Dokument Projektplan. Für jede Iteration wird der Projektplan analog zum Exceldokument erweitert.
Verantwortlich	ameier
Abhängigkeiten	-

5.1.3 Excel Zeiterfassung	5.5 Std.
Beschreibung	Die persönliche Zeiterfassung für das Projekt.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.1.4 Review Projektplan	5 Std.
Beschreibung	Review des Projektplans Excel und Word.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.1.1 Projektplan Excel 5.1.2 Projektplan Dokument

3.3.3.2 REQUIREMENTS

5.2.1 Anforderungsspezifikation	11 Std.
Beschreibung	Erweitern der bestehenden Anforderungsspezifikation mit funktionalen und nicht funktionalen Anforderungen, sowie Integration der Use Cases.
Verantwortlich	jzimmerm
Abhängigkeiten	5.2.2 Use Cases

5.2.2 Use Cases	12 Std.
Beschreibung	Erweitern der Use Cases die für die Anforderungsspezifikation von belangen sind.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.2.3 Review Anforderungsspezifikation	2 Std.
Beschreibung	Review der Anforderungsspezifikation
Verantwortlich	ameier
Abhängigkeiten	5.2.1 Use Cases

3.3.3.3 ANALYSE

5.3.1 Domainanalyse Mobile WMS Client	8 Std.
Beschreibung	Erarbeitung der Domainanalyse bezüglich des Mobile WMS Clients.
Verantwortlich	ameier
Abhängigkeiten	-

5.3.2 Review Domainanalyse Mobile WMS Client	2 Std.
Beschreibung	Review der Domainanalyse
Verantwortlich	jzimmerm
Abhängigkeiten	5.3.1 Domainanalyse

5.3.3 SSD + Contracts	8 Std.
Beschreibung	Erweiterung der SSDs anhand neuer Anforderungsspezifikation (Server). Beschreibung gewisser SSD anhand der Anforderungsspezifikation (Mobile). Beschreibung von Contracts bezüglich der Domainanalyse (Mobile).
Verantwortlich	jzimmerm
Abhängigkeiten	5.2.1 Anforderungsspezifikation 5.3.1 Domainanalyse

5.3.4 Mobile Evaluierung Android WMS Viewer	8 Std.
Beschreibung	Evaluierung einer WMS Library für Android.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.3.1 Domainanalyse

5.3.5 Tomcat 6.x Funktionsüberprüfung	1 Std.
Beschreibung	-
Verantwortlich	ameier
Abhängigkeiten	-

5.3.6 Definition der Implementationsarbeiten Mobile WMS Client	2 Std.
Beschreibung	-
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.3.1 Domainanalyse

3.3.3.4 DESIGN ANALYSE

5.4.1 Softwarearchitektur Dokument	36 Std.
Beschreibung	Die Softwarearchitektur wird in einem Dokument erfasst. Dazu gehören unter anderem die Erklärung der Packetstruktur, der Klassen und aussagekräftige Sequenzdiagramme. Das bestehende Dokument wird erweitert.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.4.2 Prototyp Configfile Relationhandling	8 Std.
Beschreibung	Definition des XSD für die Konfiguration von “echten“ Relationen.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.2.1 Anforderungsspezifikation

5.4.3 Prototyp Mobile WMS Client	20 Std.
Beschreibung	Erste Installation der evaluierten Library und Anzeige von Kartenmaterial via WMS Abfrage.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.3.4 Evaluation Library für Mobile WMS Client

5.4.4 Software Architektur Review	8 Std.
Beschreibung	-
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.4.1 Softwarearchitektur Dokument

3.3.3.5 IMPLEMENTATION

5.5.1 Relationhandling	84 Std.
Beschreibung	Erweiterung des Configfiles zur Unterstützung “echter“ Relations. Implementation der Unterstützung für den Import “echter“ Relations. Konsistenzprüfung der Relations Configfile zu Datenbank.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.5.2 Differential Update	36 Std.
Beschreibung	Implementierung des Differential Update
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.5.3 Maskierungslayer für Showcase (Clipping-Problem)	10 Std.
Beschreibung	Erstellen eines Maskierungslayers (Boundary) zum überdecken von unerwünschtem Kartenmaterial
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.5.4 GeoServer: Render Fehler	5 Std.
Beschreibung	Eruierung des Renderfehlers (Fehlende Anzeige von Gebäuden, Strassen, etc.)
Verantwortlich	ameier
Abhängigkeiten	-

5.5.5 GeoServer: Out of Memory Fehler	4 Std.
Beschreibung	Eruierung des Out of Memory Fehler.
Verantwortlich	ameier
Abhängigkeiten	-

5.5.6 GeoServer: Aufsetzen / Tiling / Cashing	10 Std.
Beschreibung	Fedora 12 Server mit Tomcat 6, pgsql und GeoServer 2.0 aufsetzen. Aktivieren und Testen der Cashingfunktionalität des GeoServers.
Verantwortlich	ameier
Abhängigkeiten	-

5.5.7 Unittests	22 Std.
Beschreibung	Erstellen von Unittests vorhandene und neue Funktionalität
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.5.8 Konsistenzprüfung GeoServer 2.0	33 Std.
Beschreibung	Anpassung an die geänderte Ordnerstruktur im GeoServer 2.0.1
Verantwortlich	ameier
Abhängigkeiten	-

5.5.9 Allgemeines Refactoring	33 Std.
Beschreibung	
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.5.10 Showcase Websites anpassen	6 Std.
Beschreibung	Erweiterung der Suchfunktionalität für die neuen Relations, Anpassungen MapCompare.
Verantwortlich	ameier
Abhängigkeiten	-

5.5.11 Funktionalitätstests Windows	17 Std.
Beschreibung	Kompletter Servertest
Verantwortlich	ameier
Abhängigkeiten	-

5.5.12 Funktionalitätstests Unix	23 Std.
Beschreibung	Kompletter Servertest
Verantwortlich	ameier
Abhängigkeiten	-

5.5.13 Installer	2 Std.
Beschreibung	Anpassungen am Installer
Verantwortlich	ameier
Abhängigkeiten	-

5.5.14 IndoorGuide4Android	10 Std.
Beschreibung	Zur Verfügung stellen einer OSM-in-a-Box Umgebung für die Bachelorarbeit IndoorGuide4Android.
Verantwortlich	ameier, jzimmerm
Abhängigkeiten	-

5.5.15 Anpassung SLD GeoServer 2.0	50 Std.
Beschreibung	Anpassen und neu erstellen der SLD des GeoServer 2.0
Verantwortlich	ameier
Abhängigkeiten	-

5.5.16 WMS Konfiguration GeoServer und GeoWebCache	20 Std.
Beschreibung	Richtige konfiguration des WMS services des GeoServer und des GeoWebCache
Verantwortlich	ameier
Abhängigkeiten	-

5.5.17 Installer	10 Std.
Beschreibung	Installer für Windows und Unix anpassen.
Verantwortlich	ameier
Abhängigkeiten	-

3.3.3.6 QUALITÄTSMASSNAHMEN

5.6.1 Technikstudium	37 Std.
Beschreibung	Einlesen in die verschiedenen zu verwendenden Technologien.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.6.2 Risikomanagementdokument	1 Std.
Beschreibung	Erstellen des Risikomanagement.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.6.3 Codereview	4 Std.
Beschreibung	Review des Codes
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

3.3.3.7 DOKUMENTATION

5.7.1 Abstract	2 Std.
Beschreibung	Abstract
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.7.2	2 Std.
Beschreibung	Broschüre
Verantwortlich	ameier
Abhängigkeiten	-

5.7.3	2 Std.
Beschreibung	Management Summary
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.7.4 Technischer Bericht	2 Std.
Beschreibung	Technischer Bericht
Verantwortlich	jzimmem
Abhängigkeiten	-

5.7.6 Benutzer und Installationshanbuch	12 Std.
Beschreibung	Benutzer- und Installationshanbuch
Verantwortlich	ameier
Abhängigkeiten	-

5.7.8 Plakat	2 Std.
Beschreibung	Plakat
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

5.7.9 Gesamtdokumentation	40 Std.
Beschreibung	Gesamtdokumentation
Verantwortlich	ameier/jzimmerm

Abhängigkeiten	-

3.3.3.8 SITZUNGEN

5.8.1 Wöchentliche Sitzungen	45 Std.
Bescahreibung	1.5 h pro Woche
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.8.2 Sitzungsvorbereitungsprotokoll	7 Std.
Beschreibung	Sitzungsvorbereitungsprotokoll erstellen. Das jeden Mittwoch Abend Herr Keller gesendet wird.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.8.1 Sitzungen

5.8.3 Sitzungsprotokoll	7.5 Std.
Beschreibung	Sitzungsprotokoll erstellen.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.8.1 Sitzungen

5.8.4 Sitzungsprotokoll Review	7.5 Std.
Beschreibung	Review des Protokolls.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.8.3 Sitzungsprotokoll

3.3.4 RISIKO MANAGEMENT

Risiko Analyse
Projektname: OpenStreetMap-in-a-Box 1.0
Projektmanager: Meier Andreas, Zimmermann Joram
Datum Kalkulation: 03.03.2010

	Risiko Bewertungen							
Risk ID	**Risiko**	**Auswirkung**	**Massnahme**	**Kosten der Massnahmen in h**	**Max. Schaden in h**	**Wahrschein - lichkeit des Eintreffens**	**Gewichteter Schaden in h**	**Priorität**
R01	**Technikstudium höher als erwartet**	**Weniger Zeit für die Implementation**	**Mehr Zeit investieren**	**40**	80	15%	**12**	**Tief**
R02	**Aufgabenstellung unklar definiert**	**Es ensteht mehr Stundenaufwand**	**Dokumentation und Aufgabenstellung anpassen**	**30**	100	5%	**5**	**Mittel**
R03	**Hardware-Ausfall**	**Arbeitsunfähigkeit während Ausfall**	**Hardware ersetzen**	**10**	42	10%	**4**	**Mittel**
R04	**Grundlegende Änderung der OSM Daten**	**osm2gis funktioniert nicht mehr**	**Rewriting des Parsers**	**100**	400	1%	**4**	**Hoch**
R05	**Probleme bei Fedora Installation**	**Showcase nicht lauffähig**	**Mehr Zeit investieren**	**40**	100	50%	**50**	**Hoch**
R06	**Zeitunterschätzung Differential update**	**Diff update ist nicht lauffähig**	**Mehr Zeit investieren**	**20**	50	40%	**20**	**Hoch**
R07	**Zeitunterschätzung Relationhandling**	**Relationhandling funktioniert nicht**	**Mehr Zeit investieren**	**40**	100	30%	**30**	**Hoch**
	Total Kosten in Arbeitspaketen enthalten			**280**				
	Total Rückstellungen						**125**	

3.3.5 INFRASTRUKTUR

Als Infrastruktur dienen uns folgende Komponenten:

- Linux Server:
  - Tbd.
- Ein Developerwiki:
  - http://dev.ifs.hsr.ch/osminabox/
- SVN-Repository:
  - http://sifsv002.hsr.ch/svn/osminabox/
- Daily-Build mit Ant.
  - http://dev.ifs.hsr.ch:8080/

3.3.5.1 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.

3.3.5.2 ENTWICKLUNGSSOFTWARE VERSION

Die Entwicklung wird mit folgender Software durchgeführt:

- Eclipse Ganymede
- Java JDK 1.6
- PostgreSQL 8.4
- PostGIS 1.5
- Apache Tomcat 6
- Geoserver 2.0

3.3.6 QUALITÄTSMASSNAHMEN

3.3.6.1 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)

3.3.6.2 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.

3.3.6.3 TESTPLANUNG

3.3.6.3.1 FUNKTIONALE TESTS

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 Testszenarien ist der Entwickler des entsprechenden Programmteils selbst verantwortlich.

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

3.4 PROJEKTMONITORING

3.4.1 SOLL-IST-ZEIT-VERGLEICH

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

Überpakete	Soll	Ist	Differenz
Projekt Management	32.5	22.0	10.5
Requirements	25.0	4.0	21.0
Analyse	29.0	18.0	11.0
Design Analyse	72.0	42.5	29.5
Implementation	390.0	472.0	-82.0
Qualitätsmassnahmen	42.0	58.0	-16.0
Dokumentation	62.0	68.0	-6
Sitzungen	67.0	67.0	0

3.4.2 CODESTATISTIK

Die folgenden Codestatistiken wurden mit Structure101 generiert:

Size	
Jars (and / or classpath directories)	1
Packagages (that contain classes)	41
Classes	303
NI(Number of bytecode Instructions)	32’000
LOC(Non Comment Non Blank Lines of Code)	14’000

Term	Threshold	#Offenders	Offenses (%)	XS contribution
Tangled (design)	0	4 of 12	33%	79%
Fat (design)	120	0 of 12	0%	0%
Fat (leaf package)	120	1 of 41	2%	21%
Fat (class)	120	0 of 303	0%	0%
Fat (method)	15	0 of 0	0%	0%
Total				100%

3.4.3 SITZUNGSPROTOKOLLE

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

3.5 USERMANUAL

3.5.1 INSTALLATION

3.5.1.1 PRECONDITIONS

1. Java 1.6 installed
2. Tomcat 6.0 installed (Download from http://tomcat.apache.org/)
3. PostgreSQL 8.4 installed (Download from http://www.postgresql.org/)
4. PostGIS 1.5.0 installed (Download from http://postgis.refractions.net/)
5. Tomcat has to be running
6. The running OSM-in-a-Box has to be conntected to the internet.
7. Download OpenStreetMap-in-a-Box for windows or unix (Download from http://dev.ifs.hsr.ch/releases/osminabox/).

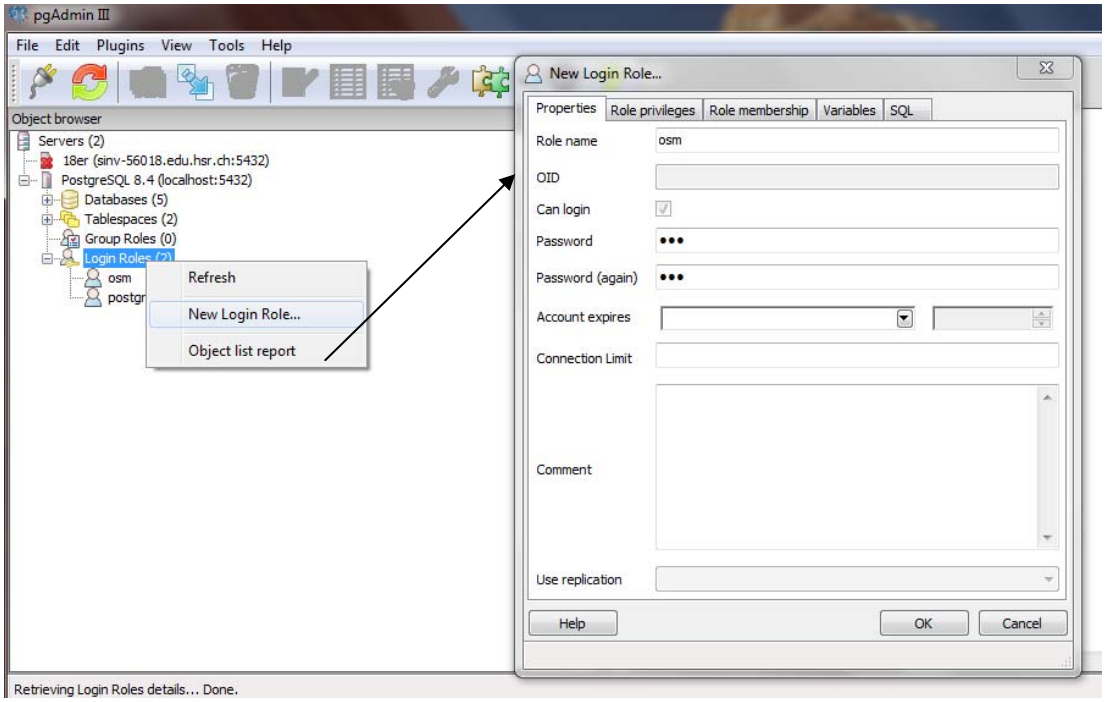
If you want to install OSM-in-a-Box on a Fedora 12 system, please refer to the Install_OSMinaBox_on_Fedora12 document to meet the preconditions.

3.5.1.2 STEP 1 CREATE A DATABASE

Create a database with the PostGIS template.

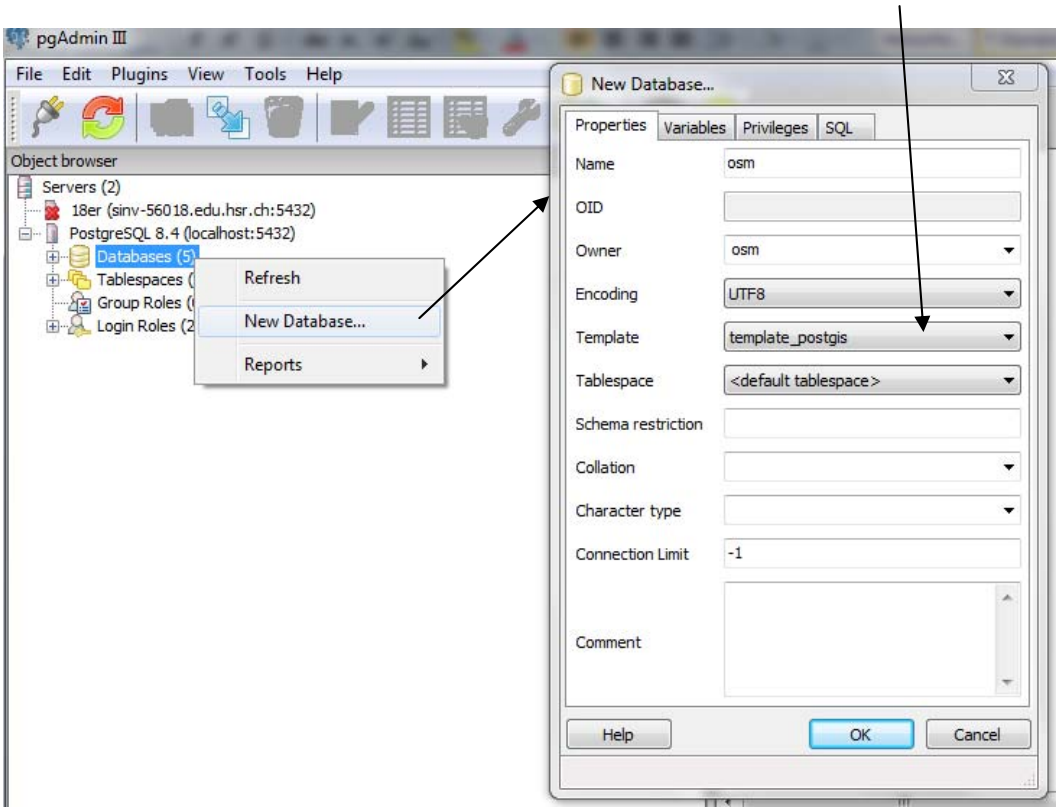
3.5.1.2.1 ON WINDOWS SYSTEMS

1. Open pgAdmin.
2. Connect to the local PostgreSQL-Server.
3. Create a new Role.

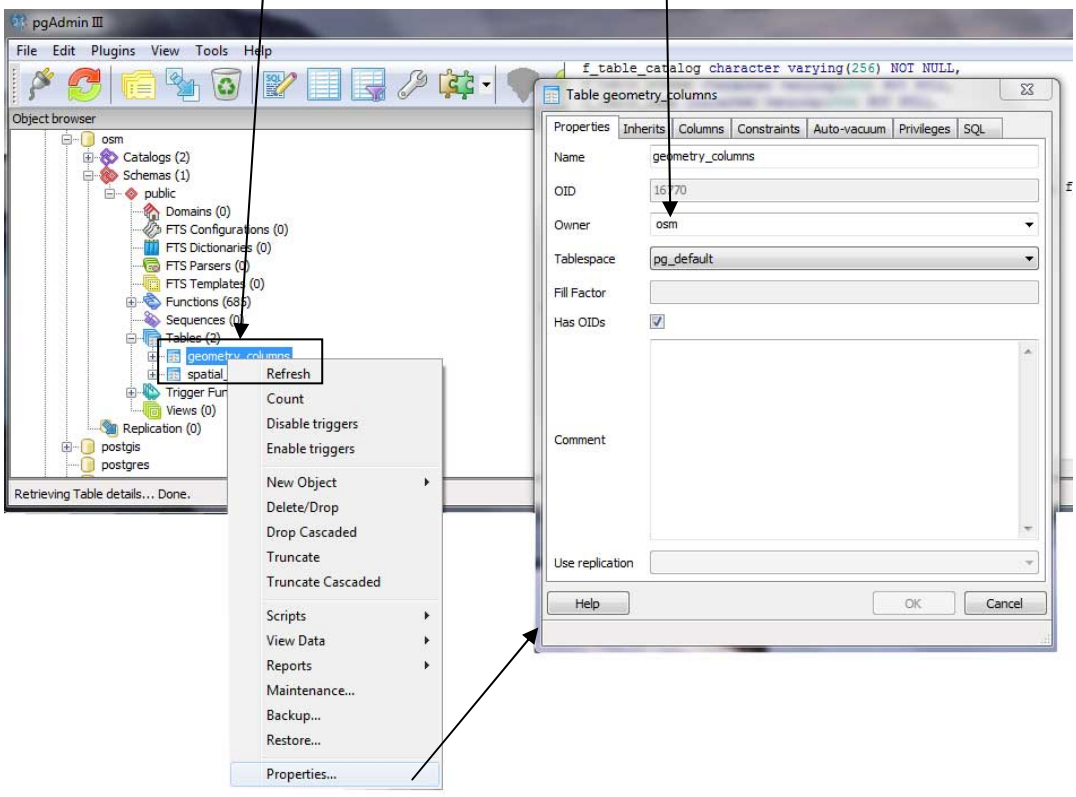


4. Figure 50: Creating a new role

5. Create a new database with the owner you created before and use the PostGIS template!



6. Make sure the 2 PostGIS template tables has the owner of the database you created!



3.5.1.2.2 ON UNIX SYSTEMS

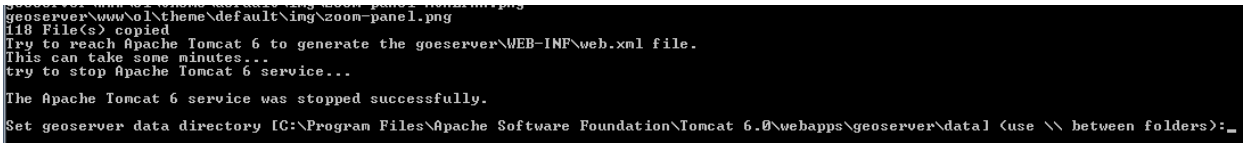
- createdb -Uosm osm
- createlang -Uosm plpgsql osm
- psql -Uosm -d osm -f /usr/share/pgsql/contrib/postgis.sql
- psql -Uosm -d osm -f /usr/share/pgsql/contrib/spatial_ref_sys.sql
- psql -Uosm -d osm -f /usr/share/pgsql/contrib/postgis_comments.sql
- psql -Uosm -d osm -f /usr/share/pgsql/contrib/hstore.sql

3.5.1.3 STEP 2 USE INSTALLER

1. Be sure that your tomcat service is running!
2. If you have downloaded the OpenStreetMap-in-a-Box 2.0 then you can unpack it.
3. Navigate to the directory you unpacked the zip.
4. 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+xinstaller.sh
        ii. Command to run installer sh: *sh –x installer.sh*
5. Set the folder where osm2gis should be installed:

6. Set the tomcat webapps folder:

7. Set the geoserver data directory:



8. Now you have to set several configuration informations:
    a. The password for the geoserver admin account
    b. The db name of the db you created before.
    c. The db user you created before.
    d. The db password for the user.
    e. The bounding box informations for the GeoServer and osm2gis.
        i. Longitude min x [-180.0 for whole world]
        ii. Longitude max x [180.0 for whole world]
        iii. Latitude min y [-90.0 for whole world]
        iv. Latitude max y [90.0 for whole world]

```
Set geoserver admin password [geoserver]:
Set db name:osm
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\catalog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set db user:osm
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\catalog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set db password:osm
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\catalog.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set boundingbox longitude min x [-180.0 for whole world]:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\services.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set boundingbox longitude max x [180.0 for whole world]:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\services.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set boundingbox latitude min y [-90.0 for whole world]:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\services.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
Set boundingbox latitude max y [90.0 for whole world]:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\data\services.xml
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
C:\Program Files\osm2gis\config\osm2gis.properties
1 File(s) copied
C:\\\\Program Files\\\\Apache Software Foundation\\\\Tomcat 6.0\\\\webapps
The Apache Tomcat 6 service is starting.
The Apache Tomcat 6 service was started successfully.

You succesfully installed OSM-in-a-Box
"Don't forget to set you personalised information on the Geoserver, under configuration-->server"
Next step is to do an --initial-import with the osm2gis
if you have done this you can go to the url: http://localhost:8080/osm2gisdemo/ to view your map!
Press any key to continue . . .
```

9. After that you have successfully installed the OpenStreetMap-in-a-Box.
10. Next step is to do an initial import more information on that see chapter 0.

3.5.2 MAPPING CONFIGURATION

When installing the OSM-in-a-Box 1.0 application, a default Schema Mapping File is distributed. It's located in the installation directory config/mappingconfig.xml.

The mappingconfig.xml file contains information about which tables need to be created (incl. columns) as well as user defined sql statements for creating needed views etc.

It also contains a mapping section. In the mapping are rules defined for mapping OSM entities to their designated tables.

The standard mapping configuration of the OpenStreetMap-in-a-Box will be automatically used if you do not specify another xml file.

If you made changes to the Schema Mapping File you can use the consistency check to check your installation, more information about this see chapter 3.5.5.

3.5.2.1 BASE SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<schema_def_and_mapping
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="mappingconfig.xsd">

    <dst_schema_def>

    </dst_schema_def>

    <src_to_dst_mappings>

    </src_to_dst_mappings>

</schema_def_and_mapping>
```

Figure 51: Base schema of mapping configuration

3.5.2.2 CONFIGURING DESTINATION SCHEMA (<DST_SCHEMA_DEF>)

In the <dst_schema_def> xml tag you can configure tables, join tables, views and user defined data.

```
<dst_schema_def>
    <dst_table_def name="gisentity">
        <!-- Definition of the table -->
    </dst_table_def>

    <dst_table_def_user_defined>
        <!-- User defined SQL statements -->
        <![CDATA[
            CREATE INDEX "water_osm_id_idx" ON "water" USING btree ("osm_id");
        ]]>
    </dst_table_def_user_defined>

    <dst_join_table_def name="network_to_trainroute">
        <!-- Definition of relation table -->
    </dst_join_table_def>

    <dst_view_def name="landuse_lookup">
        <!-- Definition of the view -->
        <![CDATA[

        ]]>
    </dst_view_def>
</dst_schema_def>
```

Figure 52: Configuring destination schema

3.5.2.2.1 DEFINING TABLES <DST_TABLE_DEF>

By defining a table you must define the tablename in *<dst_table_def name="tablename">*

Mandatory columns in every table are

- id
- osm_id
- geom (not needed for real database relation tables)

```
<dst_table_def name="gisentity">
    <dst_column name="osm_id" type="bigint" not-null="true" />
    <dst_column name="lastchange" type="TIMESTAMP" not-null="false" />
    <dst_column name="typ" type="VARCHAR(30)" not-null="false" />
    <dst_column name="name" type="VARCHAR(255)" not-null="false" />
    <dst_column name="keyvalue" type="hstore" />
</dst_table_def>
<dst_table_def name="water" inherits="gisentity">
    <dst_column name="id" type="serial" primary-key="true" />
    <dst_column name="geom" type="geometry(4326, 'MULTIPOLYGON', 2)" />
</dst_table_def>
```

Figure 53: Defining tables

3.5.2.2.1.1 THE GEOMETRY DATATYPE

Every table which stores points, linestrings or multipolygons must have a geometry type. This can be declared with *<dst_column name="geom" type="geometry(4326, 'MULTIPOLYGON', 2)".* With this information the osm2gis will generate the SQL *Select AddGeometryColumn('tablename', 'geom', 4326, 'MULTIPOLYGON', 2);*

Depending on the information you want to store in this table, the geom type must be chosen:

- POINT for inserting Nodes (pois, place, etc.)
- LINESTRING for inserting WAYS (roads, waterways, etc.)
- MULTIPOLYGON for inserting RELATIONS / CLOSED WAYS (forest, water, etc.)
  - Since only Multipolygon- /Boundary-Type-Relations are supported, the common naming for these Relations and closed ways would be "Areas".
- Relations of another type than multipolygon or boundary have no geom column since they reference their members in other tables. Using Views and Joins the geom data of the referenced members are used to satisfy the GeoServers needs.

3.5.2.2.2 DEFINING USER DEFINED SQL <DST_TABLE_DEF_USER_DEFINED>

For creating user defined sql use the <dst_table_def_user_defined> tag.

```
<dst_table_def_user_defined>
  <![CDATA[
    CREATE SEQUENCE pois_myserial_seq;
  ]]>
</dst_table_def_user_defined>
```

Figure 54: Defining user defined SQL

3.5.2.2.3 DEFINING JOIN TABLES FOR N:N RELATIONS

OSM-in-a-Box 1.0 is able to map "real" relations from OSM Relation into a Database N:N relation. You define it as follow:

```
<dst_join_table_def name="trainroute_to_railway">
    <dst_column name="trainroute_id" type="int" primary-key="true" references="trainroute" />
    <dst_column name="railway_id" type="int" primary-key="true" references="railway" />
</dst_join_table_def>

<dst_join_table_def name="trainroute_to_railwaystation">
    <dst_column name="trainroute_id" type="int" primary-key="true" references="trainroute" />
    <dst_column name="railwaystation_id" type="int" primary-key="true" references="railwaystation" />
    <dst_column name="role" type="VARCHAR(255)" />
</dst_join_table_def>
```

You have to enter a name of the table, we use the "_to_" to describe that it is a join-table. Everything is equal to defining a table (see chapter 3.5.2.2.1) except the references tag. Here you have to define on which table this column references. Dst_column with a references tag must have primary-key = "true"!

You also can define different columns for adding more information to the relation entry.

3.5.2.2.4 DEFINING VIEWS <DST_VIEW_DEF>

Views can only be defined in the end of the <dst_schema_def> section.

On the attribute name of dst_view_def, you have to declare the name of the view.

```
<dst_view_def name="landuse_lookup">
    <![CDATA[
        CREATE VIEW landuse_lookup AS
            SELECT First_Select.osm_id,
               First_Select.name,
               First_Select.typ,
               landuse.geom
            FROM landuse,
            (SELECT MAX(osm_id) AS osm_id, name, typ
            FROM landuse
            WHERE name!='' AND typ = 'glacier'
            GROUP BY name, typ) as First_Select
            WHERE First_Select.osm_id = landuse.osm_id;

            INSERT INTO geometry_columns(
                        f_table_catalog, f_table_schema, f_table_name, f_geometry_column, coord_dimension, srid, "type")
            VALUES ('', 'public', 'landuse_lookup', 'geom', 2, 4326, 'MULTIPOLYGON');
    ]]>
</dst_view_def>
```

Figure 55: Defining views

If you define views, you have to make an insert into geometry_columns for registering that view so the GeoServer know this view.

3.5.2.3 CONFIGURING SOURCE SCHEMA MAPPING

There are 4 different type of mappings (Point, Linestring, Multipolygon and Relation). Point, Linestring and Multipolygon are configured equal only the type, of the mapping element type, is different to the type of the mapping you want to use. This configuration is shown in chapter 3.5.2.3.1.

In the Mappingtype Relation exists three equal tags <and_ed_conditions>, <dst_table> and <dst_columns> Mappingtype is extended by a new tag <members>. This configuration is shown in chapter 3.5.2.3.2.

3.5.2.3.1 DEFINE POINT, LINESTRING AND MULTIPOLYGON MAPPINGS

Now you can define mappings on your tables you have defined in chapter 2.2.

In a mapping-tag there exists an <and_ed_conditions>-tag. Here are conditions stored which an OSM entity (Node, Way, Relation) must fulfill to be inserted in the destination table specified in this mapping-tag. Since they are AND-ED-conditions, all of the following <nodes_tags/>-tags must be contained by an OSM entities Tag-tag collection. A <nodes_tags k=”key” v=”value”/>-tag is equal to a <tag k=”key” v=”value”/>-tag from an OSM entities Tag-tag-collection.

An OR-ED-condition can be achieved by creating multiple <mapping>….</mapping>-sections with the same destination table.

In the <dst_table> you configure the destionation table you defined in chapter 2.2. Only one dst_table-tag can occur within one <mapping>-section.

After you defined the destination table, you need to define which value comes into which column.

Please note that type can be point, linestring or multipolygon.

```
<mapping type="point">
    <and_ed_conditions>
        <tag k="place" v="region" />
    </and_ed_conditions>
    <dst_table name="place" />
    <dst_columns>
        <column name="osm_id" value="%attribute_id%" />
        <column name="lastchange" value="%attribute_timestamp%" />
        <column name="type" value="region" />
        <column name="name" value="%tag_name%" />
        <column name="population" value="%tag_population%" />
        <column name="keyvalue" value="%tags_all%" />
        <column name="geom" value="%geom%" />
    </dst_columns>
</mapping>
```

Figure 56: Point, Linestring and Multipolygon mapping configuration

3.5.2.3.1.1 POSSIBLE VALUES IN A COLUMN

First of all: make sure that whatever value will be inserted into a column, since all values come from an xml file and are Strings, they need to be casted to the destination column datatype. If this is not possible, NULL will be inserted. If the column doesn’t allow NULL-values, the whole OSM entity will be skipped.

The simplest way of inserting a value is using a constant value for this mapping section. This would look like this: <column name=”type” value=”hotel”/>

This makes sense if you are using one destination table in multiple mappings and you want to insert a different value according to the mapping section.

If you want to insert a value which comes from the entity's tag or one of its subtag, there are the following "variables" to use inside the value="" attribute of a column-tag:

- %attribute_xxx% where xxx is any key from an entities main tag
- %tag_xxx% where xxx is any key from an entities tag-subtag
- %tags_all% for inserting all key-value-pairs from the tags-collection of a node into a column with datatype HStore
- %nd_all% for inserting all referenced Node-Ids from a Way as a concatenated String
- %members_all% for inserting all referenced Way-Ids from a Relation as a concatenated String
- %geom% for inserting an entities geometry values, mandatory column in every destination table

%attribute_xxx% example

<node id="123" timestamp="1.1.2002…" lat="47" lon="8" …/>

If you want the node's id in your osm_id column, the column-tag in your mapping section would look like this:
<column name="osm_id" value="%attribute_id%"/>

%tag_xxx% example

```
<node id="123" timestamp="1.1.2002…" lat="47" lon="8" …/>
    <tag k="name" v="Hilton"/>
</node>
```

If you want the value „Hilton" from the tag with the key „name" in your destination column „name", this would look like this:
<column name="name" value="%tag_name%"/>

%tags_all% example

It's usefull to have a column for all tags an entity comes with if you want to "back-up" this information. This can be achieved with the datatype HStore and the %tags_all% variable.

```
<node id="123" timestamp="1.1.2002…" lat="47" lon="8" …/>
    <tag k="name" v="Hilton"/>
    <tag k="created_by" v="JOSM"/>
    <tag k="is_in" v="Miami"/>
</node>
```

If you want to save all this node's tags-collection into an associative HStore datatype column, the column-tag would look like this:
<column name="keyvalue" value="%tags_all%"/>

%geom% example

Since every table needs a Geometry datatype column for storing latitude / longitude data, the value for this column can be extracted by the %geom% variable.

<node id=”123” timestamp=”1.1.2002…” lat=”47” lon=”8” …/>
<tag k=”name” v=”Hilton”/>
</node>

Which Geometry-Type the column has was already defined in the <dst_table_def … />-tag. The appropriate column definition in the mapping section would look like this:
<column name=”geom.” value=”%geom%”/>

3.5.2.3.2 DEFINE RELATION MAPPINGS

For the case to map OSM relations it is possible to define relation mappings. In chapter 3.5.2.2.3 it is described how to define a join table. Now this join tables have to be mapped to the OSM data.

```
<!-- NETWORK -->
<mapping type="relation">
    <and_ed_conditions>
        <tag k="type" v="network" />
    </and_ed_conditions>
    <dst_table name="network" />
    <dst_columns>
        <column name="osm_id" value="%attribute_id%" />
        <column name="lastchange" value="%attribute_timestamp%" />
        <column name="type" value="%tag_type%" />
        <column name="operator" value="%tag_operator%" />
        <column name="name" value="%tag_name%" />
        <column name="keyvalue" value="%tags_all%" />
    </dst_columns>
    <members all_required="false">
        <related_table name="route">
            <join_table name="network_to_route" />
            <join_table_columns>
                <column name="network_id" value="%db_relation_id%" />
                <column name="route_id" value="%db_member_id%" />
                <column name="role" value="%member_role%" />
            </join_table_columns>
        </related_table>
        <related_table name="trainroute">
            <join_table name="network_to_trainroute" />
            <join_table_columns>
                <column name="network_id" value="%db_relation_id%" />
                <column name="trainroute_id" value="%db_member_id%" />
                <column name="role" value="%member_role%" />
            </join_table_columns>
        </related_table>
    </members>
</mapping>
```

Figure 57: Relation mapping configuration of relations

The <and_ed_conditions>, <dst_table> and <dst_columns> configuration is equal to chapter 3.5.2.3.1.

For creating real database relations between different table, a predefined join table is required. In the <members> tag all tables, in which any of a relations member can be found, are listed. Make sure that a separate mapping exists to map a member (for example a railway) in its table. Depending on the related table where a member is found, the corresponding join table will be filled with the primary key of the relation and the primary key of the referenced member. Additionally, if such a member contains a role attribute, it can be added to that join table entry. The following parameters of are fixed:

- **%db_relation_id%** - The primary key of the relation from its mapped table
- **%db_membe_id%** - The primary key of the member from the related table
- **%member_role%** - The role attribute from the xml file belonging to this member (optional)

3.5.3 INITIAL IMPORT

The import can be executed using the following options:

3.5.3.1 IMPORT A PLANET FILE FROM AN URL

osm2gis --initial-import -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.

3.5.3.2 IMPORT THE PLANET FILE FROM A LOCAL PATH

osm2gis --initial-import -f [planetfile]

Additional commands:

-m [mappingconfig file]
-h [DB host]
-d [Database]
-u [DB username]
-p [DB password]

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

3.5.3.3 IMPORT THE PLANET FILE DATA USING A BOUNDING BOX

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

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

3.5.3.4 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 bzcat utility to decompress the planet file and pipe the resulting stream directly into the standard input of the osm2gis application:

bzcat [planetfile] | osm2gis –initial-import

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

Important: bzcat is a linux/unix tool. Under windows the cygwin implementation can be used

3.5.4 DIFFERENTIAL UPDATE

Information about OpenStreetMap differential update is available on
http://wiki.openstreetmap.org/wiki/Planet.osm/diffs.

3.5.4.1 SCHEDULED UPDATE

A scheduled update will run continuously by itself, keeping the map data up to date. According to the frequency, osm2gis will download and import the newest files or, fetch up to the current if multiple files are available.

osm2gis --schedule-update

Required for first use:

--frequency [daily|hourly|minutely]
--initial-diff [Name of the Initial Diff File for daily updates]
--initial-diff-replicate [Name of the Initial Diff File for minutely and hourly updates]

Additional commands:

-m [mappingconfig file]
-h [DB host]
-d [Database]
-u [DB username]
-p [DB password]
--latmin [minimum latitude]
--lonmin [minimum lonitude]
--latmax [maximum latitude]
--lonmin [maximum longitude]

3.5.4.2 NON SCHEDULED UPDATE

The non scheduled update will only run once for the given file.

osm2gis --update

Required for first use:

--frequency [daily|hourly|minutely]
--initial-diff [Name of the Initial Diff File for daily updates]
--initial-diff-replicate [Name of the Initial Diff File for minutely and hourly updates]

Additional commands:

-m [mappingconfig file]
-h [DB host]
-d [Database]
-u [DB username]
-p [DB password]
--latmin [minimum latitude]
--lonmin [minimum lonitude]
--latmax [maximum latitude]
--lonmin [maximum longitude]

3.5.5 CONSISTENCY CHECK

If you want to check the consistency of your OpenStreetMap-in-a-Box configuration then start osm2gis with:

--consitency –m "YourSchemaMappingFile.xml"

This will write you a consistency report.

3.5.5.1 STRUCTURE OF CONSISTENCY REPORT

At first you should know the architecture of the configuration.

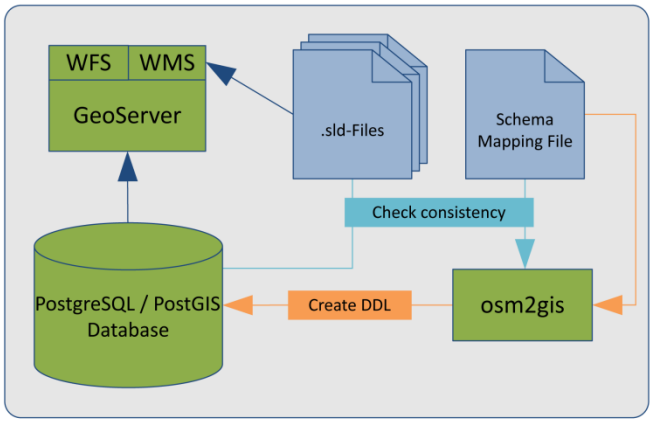


Figure 58: Configuration

3.5.5.1.1 SUMMARY

The summary shows you the count of difference(s), error(s) or hint(s) for the details.

3.5.5.1.2 DETAILS

3.5.5.1.2.1 SRC_TO_DST_MAPPINGS TO DST_SCHEMA_DEF IN MAPPINGCONFIGURATION

Here the tables and columns you defined in the mapping configuration would be checked.

```
1 Consistency check, src_to_dst_mappings to dst_schema_def in mapping configuration
-----------------------------------------------------------------
No table (buildings) found for mapping:

    <mapping type="linestring">
        <and_ed_conditions>
            <tag key="building" value="yes"/>
        </and_ed_conditions>
        <dst_table name="buildings" />
        <dst_columns>
            <column name="osm_id" value="%attribute_id%"/>
            <column name="lastchange" value="%attribute_timestamp%"/>
            <column name="type" value="building"/>
            <column name="name" value="%tag_name%"/>
            <column name="keyvalues" value="%tags_all%"/>
            <column name="geom" value="%geom%"/>
        </dst_columns>
    </mapping>
--------------------------------
-----------------------------------------------------------------
```

3.5.5.1.2.2 DST_SCHEMA_DEF IN MAPPINGCONFIGURATION TO DB

This check checks if the tables you defined in your Schema Mapping File exists in your database.

```
Consistency check, tables defined in mapping configuration to DB in use
------------------------------------------------------------------
Table: boundary
0 diff(s) found!
------------------------------------------------------------------
Table: building
0 diff(s) found!
------------------------------------------------------------------
Table: coastline
0 diff(s) found!
```

3.5.5.1.2.3 GEOSERVER FEATURE TYPE (INC. SLD) TO DB

This check checks if the GeoServer feature types exists and if the SLD-File of the feature type references to columns that doesn't exist in the DB.

```
------------------------------------------------------------------
------------------------------------------------------------------
Checking GeoServer configuration to DB
------------------------------------------------------------------
Datastore name: osm
------------------------------------------------------------------
------------------------------------------------------------------
Feature type:boundary
---------------------------------
0 error(s) found!
------------------------------------------------------------------
------------------------------------------------------------------
```

3.5.5.1.2.4 MAPPING CONFIGURATIONS TO GEOSERVER (*.SLD)

This check checks your mapping configuration against the SLD files an gives you hint on wich SLD wich mapping is not configured. These are only hints!

```
------------------------------------------------------------------
Mapping table: railwaystation
SLD-file    : C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver/data/styles/osm_railwaystation.sld
---------------------------------
Hint: No <rule> XML-tag in SLD file found, for mapping configuration:
    <mapping>
        <and_ed_conditions>
            <nodes_tags key="railway" value="halt"/>
        </and_ed_conditions>
        <dst_table name="railwaystation" />
        <dst_columns>
            <column name="osm_id" value="%node_id%"/>
            <column name="lastchange" value="%node_timestamp%"/>
            <column name="typ" value="halt"/>
            <column name="name" value="%tag_name%"/>
            <column name="keyvalue" value="%tags_all%"/>
        </dst_columns>
    </mapping>
---------------------------------
1 hint(s) found!
------------------------------------------------------------------
```

3.5.6 CREATE NEW VIEWS IN MAPPING CONFIGURATION

You can create new views in the mapping configuration as shown in chapter 2. If you just want to create these new views you can run the osm2gis with:

osm2gis –v [path to the mapping configuration file]

3.5.7 HELP

Help inside the command line can be found by typing:

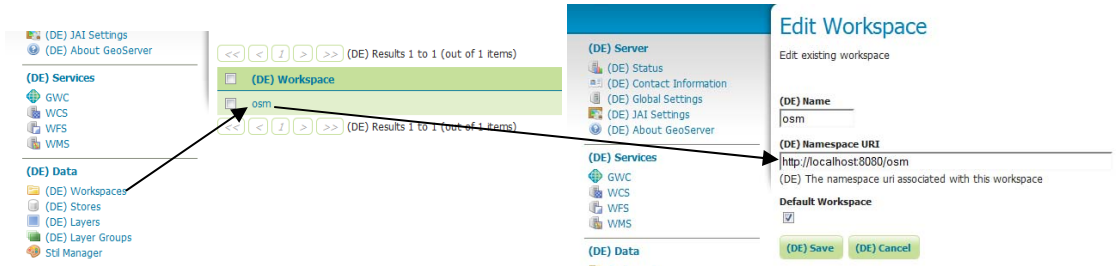
osm2gis --help

3.5.8 GEOSERVER FINE TUNING

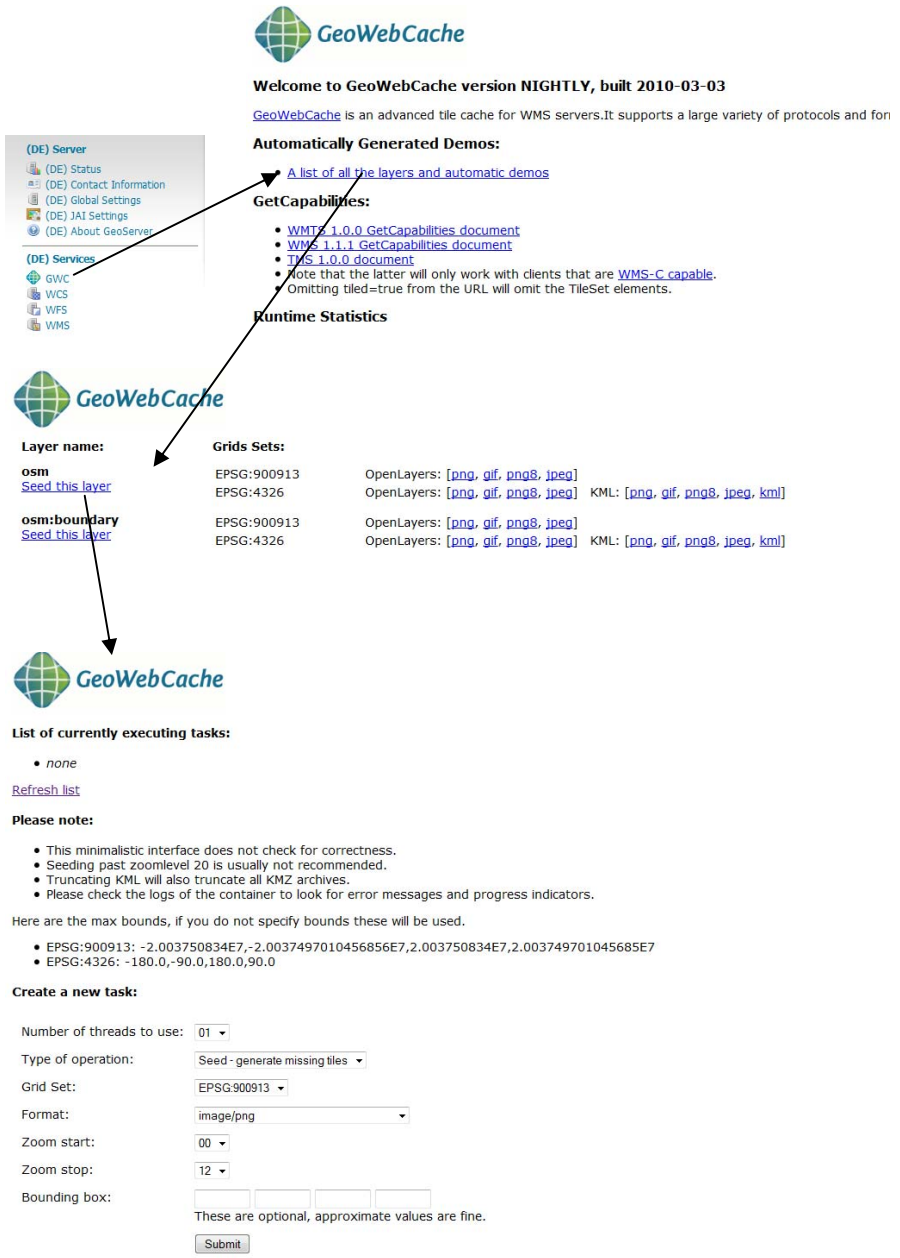
3.5.8.1 ENTER CONTACTINFORMATION

Figure 59: GeoServer Contact Information

3.5.8.2 ENTER NAMESPACE URI



3.5.8.3 SEED GEOWEBCACHE LAYERS



In this Window you can enter your bounding box and zoom layer start and end. Click on submit to start seeding.

3.6 INSTALLATION ON FEDORA 12

3.6.1 INSTALL REQUIRED PACKAGES

Install required packages

- *yum install postgresql postgresql-server postgresql-devel gcc gcc-c++ libXp libXtst make bzip2 flex*

3.6.2 JAVA JDK

Download and copy the latest version of Java 1.6 to the server (the .bin file, not the RPM).

- *cd /opt*
- *wget pathtojavajdk*

This will install Java in some subdirectory of /opt . Create a symlink to make it more userfriendly:

- *sh op/jdk-1_6_0_XX-linux-i586.bin*
- *ln -s jdk1.6.0_03 sun-java-1.6.0*

3.6.3 APACHE TOMCAT 6.0

Create Tomcat user and add Java path.

- *useradd -m tomcat*
- *echo "export JAVA_HOME=/opt/sun-java-1.6.0" >> .bashrc*
- *echo "export PATH=/opt/sun-java-1.6.0/bin:$PATH" >> .bashrc*
- *echo "export MALLOC_CHECK_=0" >> .bashrc*
- *su -l tomcat*
- *java -version*

The last command should respond with

```
java version "1.6.0_03"
Java(TM) SE Runtime Environment (build 1.6.0_03-b05)
Java HotSpot(TM) 64-Bit Server VM (build 1.6.0_03-b05, mixed mode)
```

Now installing Tomcat is as easy as grabbing the archive and unpacking it:

- *cd /opt*
- *Download from http://tomcat.apache.org with the wget command*
- *tar opt/apache-tomcat-6.0.14.tar.gz*
- *chown tomcat apache-tomcat-6.0.14 -R*
- *ln -s apache-tomcat-6.0.14 tomcat-6.0*
- *su -l tomcat*
- *cd /opt/tomcat-6.0/bin*
- *./catalina.sh start*
- *netstat -nptl*

3.6.4 CONFIGURE PROXY

Wen need a Proxyconfiguration that standard webservice httpd knows Tomcat on localhost.

- *vi /etc/httpd/conf.d/proxy_ajp.conf*

This will open the VI Editor to edit the proxy_ajp.conf. Now add this lines to this file

```
ProxyPass /geoserver/ http://localhost:8080/geoserver/
ProxyPassReverse /geoserver/ http://localhost:8080/geoserver/
```

3.6.5 CONFIGURE FIREWALL SETTINGS (IPTABLES)

To ensure that the required ports are accesible we have to configure iptables as follow:

- *vi /etc/sysconfig/iptables*

This will open the VI Editor to edit iptables. Content of this file will look like this:

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
-A RH-Firewall-1-INPUT -p icmp --icmp-type any -j ACCEPT
-A RH-Firewall-1-INPUT -p 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p 51 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp --dport 5353 -d 224.0.0.251 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -m udp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -p tcp -m tcp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 5432 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

3.6.6 INSTALL POSTGRESQL WITH POSTGIS

Fedora 12 comes with postgresql. We have to install several packages to support PostGIS and HStore.

- *yum install postgresql-devel*
- *yum install postgresql-server*
- *yum install postgis*
- *yum install postgresql-contrib*

Now we have to change the localhost in file postgresql.conf with *.

- *vi /var/lib/pgsql/data/postgresql.conf*

If you want to access postgresql Databases from remote, you have to configure pg_hba.conf

- *vi /var/lib/pgsql/data/pg_hba.conf*

Content of this file must look like this:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
local   all         all                               trust
# IPv4 local connections:
host    all         all         127.0.0.1/32          md5
host    all         all         152.96.0.0/16         md5
host    all         all         192.168.0.0/16        md5
# IPv6 local connections:
host    all         all         ::1/128               md5
```

3.6.7 CREATE DATABASE WITH POSTGIS AND HSTORE

- *createdb -Uosm osm*
- *createlang -Uosm plpgsql osm*
- *psql -Uosm -d osm -f /usr/share/pgsql/contrib/postgis.sql*
- *psql -Uosm -d osm -f /usr/share/pgsql/contrib/spatial_ref_sys.sql*
- *psql -Uosm -d osm -f /usr/share/pgsql/contrib/postgis_comments.sql*
- *psql -Uosm -d osm -f /usr/share/pgsql/contrib/hstore.sql*

3.6.8 INSTALL TRUETYPE FONT ON FEDORA 12

- *yum install rpm-build cabextract ttmkfdir wget*
- *wget http://www.mjmwired.net/resources/files/msttcore-fonts-2.0-3.spec*
- *rpmbuild -ba msttcore-fonts-2.0-3.spec*
- *yum localinstall --nogpgcheck /root/rpmbuild/RPMS/noarch/msttcore-fonts-2.0-3.noarch.rpm*

3.6.9 RESTART PREVIOUS CONFIGURED SERVICES (WHOLE SYSTEM)

To make sure everything works restart the running system.

- *shutdown –r now*

After the system is rebooted you have to start the PostreSQL, httpd and Tomcat.

Starting PostgreSQL and httpd

- */etc/init.d/postgresql start*
- */etc/init.d/httpd start*

Starting Tomcat

- *su –l tomcat*
- */opt/tomcat-6.0/bin/catalina.sh start*

3.6.10 INSTALL OSM-IN-A-BOX 1.0

Please refer to the chapter 3.5 Usermanual.

4 ANHANG

4.1 ERFAHRUNGSBERICHTE

4.1.1 ANDREAS MEIER

4.1.1.1 TEAM

Die Zusammenarbeit mit Joram Zimmermann hat sehr gut funktioniert. Da wir schon die Arbeiten in User Interfaces 1 und Software Engineering 2, sowie auch die Semesterarbeit zusammen bewältigt haben wissen wir, wie der andere denkt und wo die Stärken und Schwächen des Arbeitspartners liegen. Da diese Arbeit eine Fortsetzungsarbeit unserer Semesterarbeit ist, war von Beginn an klar, wer welche Problemstellungen lösen wird.

Durch unsere ähnlichen Stundenpläne konnten wir die Zeiten, in denen wir zusammen arbeiten mussten, gut einteilen.

Auch die Zusammenarbeit mit Herr Adrian Geiter und Christoph Egger (Bachelorarbeit IndoorGuide4Android) war unkompliziert und professionell.

4.1.1.2 ZEITPLANUNG

Der Start verlief reibungslos, wir benötigten keine Einarbeitungszeit in das Projekt, da wir uns bereits zuvor ein Semester mit diesem Thema befasst hatten. Daher konnten wir nach einer kurzen Inception und Elaboration Phase bereits mit der Implementierung beginnen.

Mein erster Teil, das Updaten des GeoServer auf Version 2, die Anpassung der Konsistenzprüfung an die neue Ordnerstruktur, sowie das Erweitern des Konsistenzchecks, verlief wie geplant.

Sehr viel Zeit benötigte ich mit der frischen Installation des Fedora 12 Showcase Server. Da ich kein Unix Anwender bin, musste ich mir alles aneignen, was man benötigt, um einen Webserver mit Tomcat und Co. zum Laufen zu bringen. Im letzten Teil konnte ich noch Zeit in Tests (osm2gis Import Konverter) sowie die Konfiguration des GeoServers stecken.

4.1.1.3 NEU ERLERNTE TECHNOLOGIEN

- OSM-Daten
- GeoServer 2.0
- GeoWebCache
- SLD styling
- Fedora 12

4.1.1.4 FAZIT

Das Ziel, eine stabile Version 1.0 von OSM-in-a-Box zu implementieren, ist uns meiner Meinung nach gelungen. Kinderkrankheiten wurden eliminiert und neue Funktionalitäten hinzugefügt.

Die Entscheidung, den mobilen Teil unserer Aufgabenstellung verkürzt mit einer Evaluierung einer vorhandenen Applikation zu ersetzen, war richtig. Somit konnte ich diese Zeit intensiv in die Stabilität des GeoServers

investieren. Positiv war auch die Betreuung durch Herrn Keller, der uns bei Fragen immer nützliche Tipps geben konnte.

4.1.2 JORAM ZIMMERMANN

4.1.2.1 TEAM

Da ich Herr Meier schon von der Berufsschule kenne, funktioniert unsere Zusammenarbeit sehr gut. Während dem Studium an der HSR haben wir bereits diverse Projekte und Miniprojekte zusammen durchgeführt (UInt1, SE2 Projekt, Semesterarbeit, etc.). In unserer Arbeitsweise sind wir sehr unterschiedlich, da wir uns jedoch gut kennen, können wir die Stärken des Anderen nutzen und seine Schwächen kompensieren.

Die Aufgabenverteilung während dem Projektverlauf war einfach, da wir sehr unterschiedliche Aufträge umzusetzen hatten und schon von der Semesterarbeit die Aufgabenbereiche verteilt waren. Da der Rewrite des Import Konverters (RelationHanlding und DifferentialUpdate) mehr Zeit in Anspruch nahm als wir eingeplant haben, verbrachte Herr Meier mehr Zeit mit der Konfiguration des GeoServers und dem Showcase.

Während der Arbeit ist das Team Geiter / Egger als praktische Anwender unseres Projektes hinzugestossen. Mit ihrer Bachelorarbeit IndoorGuide4Android benötigten sie die von uns importierten OSM Daten, vor allem die neue Funktionalität RelationHandling und DifferentialUpdate. Die Zusammenarbeit mit ihnen verlief sehr gut.

4.1.2.2 ZEITPLANUNG

Im Vergleich zur vorgängigen Semesterarbeit kannte ich mich ausgezeichnet mit dem Projekt aus. Somit benötige ich nicht viel Zeit für das Technikstudium oder dem Einarbeiten in das Aufgabengebiet. Trozdem habe ich den Aufwand der neuen Aufgabenstellung (RelationHandling und DifferentialUpdate) unterschätzt und während der Arbeit mussten Tätigkeitsbereiche neu verteilt werden. Dies funktionierte zum Glück einwandfrei und somit konnten alle gesteckten Hauptziele erreicht werden.

4.1.2.3 NEU ERLERNTE TECHNOLOGIEN

- JAXB
- OSM-Daten
- GeoServer 2.0

4.1.2.4 FAZIT

Durch das Vorwissen konnte rasch in die Implementationsphase des Projektes eingestiegen werden. Dies war nötig um die umfangreichen Verbesserungen und Erweiterungen umsetzten zu können. Auch nach etlichen Tests kamen immer wieder Fehler zum Vorschein, was auf die komplexe und inkonsistente Datenstruktur von OSM zurückzuführen ist. Nichts desto trotz ist nun eine deutlich stabilere Version entstanden, mit welcher sich sämtliche OSM Daten verarbeiten lassen.

Die Arbeit an dieser Bachelorarbeit machte viel Freude, da ich laufend einen Fortschritt sah. Dies motivierte mich sehr auch kleinere Details hinzuzufügen, um eine möglichst komplette Karte als Ergebniss zu haben.

Prof. Keller unterstütze uns sehr und scheute keine Mühe, auch komplizierte Fragen mit uns zu besprechen und nach Lösungen zu suchen.

4.2 INHALT DER CD

Der Inhalt der CD gliedert sich wie folgt:

Pfad	Dateien
/	
	Abstract_Kurzfassung_osm2gis.doc
	Gesamtdokumentation.docx
	Gesamtdokumentation.pdf
	Inhalt_CD.pdf
	Management_Summary_osm2gis.doc
	Persoenliche_Berichte.docx
	Poster_deutsch.pptx
	Technischer_Bericht.docx
/Dokumente/	
	Sämtliche Dokumente die während der Arbeit verfasst wurden, in Unterordner gegliedert. Hier befinden sich auch die Sitzungsprotokolle.
	Alle Verweise auf Dokumente aus der Gesamtdokumentation sind in diesem Verzeichnis zu finden.
/Java_doc/	
	Javadoc
/OSM-in-a-Box 1.0/	
	Installationsdateien für OSM-in-a-Box 1.0
/Source/osm2gis	
	Source des Import-Konverters osm2gis 1.0. Dieser Ordner kann im Eclipse mit Import an existing Project importiert werden.

4.3 ANHANG B GLOSSAR UND ABKÜRZUNGSVERZEICHNIS

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

WMS	*W*eb *M*ap *S*ervice: creates the tiles on GeoServer
Yum	A linux tool to install sofware from repositories.
Consistency check	Consistency check of the whole OpenStreetMap-in-a-Box configuration. Including Schema Mapping File, GeoServer and Postgres Database.

4.4 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
- [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 Geospetial 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.refractions.net/documentation/

[17] GeoWebCache documentation, http://geowebcache.org/trac

[18] stAX, http://stax.codehaus.org/Home

[19] cygwin, http://www.cygwin.com/

4.5 ANHANG D EIGENHÄNDIGKEITSERKLÄRUNG

4.5.1 ANDREAS MEIER

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:

4.5.2 JORAM ZIMMERMANN

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:

5 DOKUMENT-HISTORY

Da diese Bachelorarbeit eine Weiterführung des OpenSteetMap-in-a-Box Projektes ist, wurden gewisse Kapitel dieses Dokumentes aus der Vorgängerarbeit übernommen, abgeändert oder komplett neu geschrieben.

Eine Detaillierte Dokument-History findet man in den einzelnen Dokumenten die zu den Kapiteln gehören. Diese Dokumente findet man auf der CD.