OPENSTREETMAP-IN-A-BOX 2

GESAMTDOKUMENTATION

Gruppenmitglieder: *Meier Andreas*

Zimmermann Joram

ABSTRACT

BESCHREIBUNG

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

Im Gegensatz zu anderen Kartendiensten wie zum Beispiel Google Maps ist OSM detaillierter und aktueller. Vor allem aber können die gesammelten Daten frei genutzt werden, unter anderem in Navigationssystemen (GPS) und Smartphones.

Im Rahmen dieser Studienarbeit wurde der OSM-Konverter einer Bachelorarbeit vom Frühjahrssemester 2009 erweitert. Neu kann die interne Datenstruktur, in welche die OSM-Daten importiert werden, flexibel konfiguriert werden.

ZIELE

Für die Studienarbeit wurden folgende Ziele vereinbart:

- Rewrite des gesamten Import-Konverters (osm2gis), so dass mit einem neuen Mapping (XML) ein beliebiges Ziel-Schema und Mapping (Tabelle/ Felder/ Werte) konfiguriert werden können (ev. Preprocessing-Schritt SQL-DDL generieren).
- Refactoring, v.a. das XML Parsing der OSM-Daten.
- Falls sinnvoll Konsistenzprüfung/-erhalten des Datenschemas mit den GeoServer-Grafikkonfigurationen des Kartenservers (GeoServer).
- Integrieren der Konfiguration (GeoServer) von der Diplomarbeit HS 2009 (DA Teil B).
- Integrieren der Website (-Erweiterungen) von DA Teil B (ev. an einen definitiven Platz).

LÖSUNG

Es wurde ein XML Schema für das Schema Mapping File des osm2gis Import-Konverters erstellt. Im ersten Teil dieses XML Files wird das benutzerdefinierte Datenbankschema (Tabellen, Views) definiert. In einem zweiten Teil werden sog. Schema Mappings definiert. Schema Mapping-Einträge bestimmen, welche OSM-Daten in welche Ziel-Tabellen importiert werden. Das osm2gis-Programm musste erweitert und zu einem grossen Teil geändert werden, um die Schema Mappings und das flexiblen Ziel-Schema zu unterstützen.

Um die nicht ganz triviale Konfigurationsarbeit zu erleichtern, kann mittels Konsistenzprüfung ein Reportfile generiert werden.

Die aufbereiteten OSM-Daten werden über die Webservices eines professionellen Geo-Informationssystems (GIS) zur Verfügung gestellt.

Ein neuer Installer steht für die Inbetriebnahme von OSM-in-a-Box zur Verfügung. Der Installer enthält die GeoServer-Grafikkonfiguration einer Diplomarbeit aus dem HS 2009, sowie ein vorkonfiguriertes Schema Mapping File für die Website.

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

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

- Part I: Importer
  Development of a fully configurable importer which can import OSM files into a spatial database and hold this database up to date.
- Part II: Consistency check
  A way to check the configuration-consistency of the whole system (Schema-Mapping-File, database, GeoServer configuration).
- Part III: Geo-information server
  Configuration and installation of a geo-information server called GeoServer. Including the configuration of a Web Map Service (bitmap), a Web Feature Service (vector data, read-only) and a Web Map Tiling Service (tiled raster data).
- Part IV: Website
  Creation of a website to demonstrate the project. This website is included in the installer.
- Part V: Write access -optional-
  The write access to the create database is possible and the changes made local can be synchronized with OSM API.

SIMILAR PROJECTS

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

Tool	Description	Weblink
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

The start of the project was mostly used for technical studies on the previous work of this project, the map server (GeoServer) and the PostgreSQL database using PostGIS, an add-in for spatial data. With the big picture created, the work on the project began with the definition of the newly needed XML Schema Mapping File for a full configuration support. Next, the work was split into the rewriting and refactoring part of the import, parsing and mapping process and the newly needed part which consisted of 1. a preprocess step to automatically generate the DDL out of the XML Schema Mapping File, 2. a consistency check of the whole system and 3. an installer including a corresponding website.

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

RESULTS

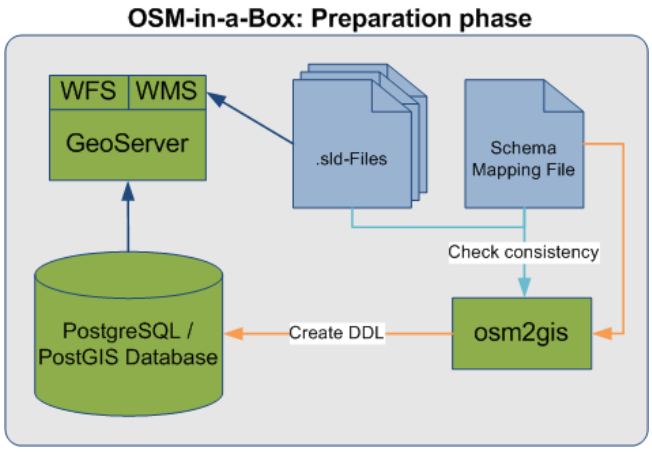


Figure 1: Schema mapping configuration

- The schema mapping file consists of a destination schema (db schema) and a mapping configuration on how the OSM data will be imported into the destination schema.
- It is possible to check the consistency of the configuration for the whole application. This includes the Schema Mapping File, the Database and the GeoServer.

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

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

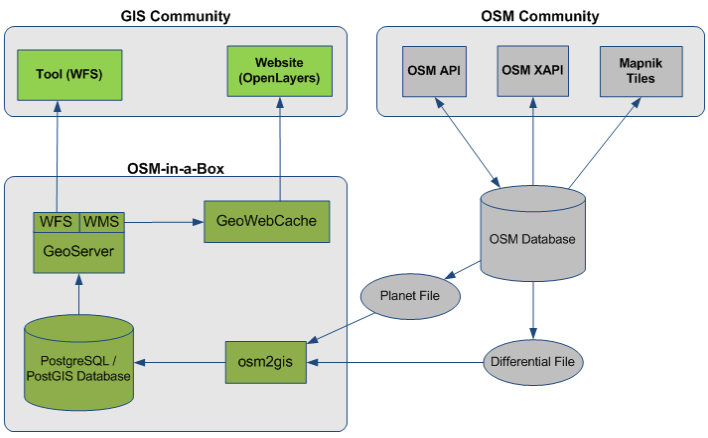


Figure 2: OSM world (right) OSM-in-a-Box and GIS world (left)

GOAL ATTAINMENT

All primary goals have been achieved. Some subordinate tasks have been added and fulfilled like the mapcompare website. The synchronization back to the OSM-API has not been achieved due to time shortage.

EXTERNAL RESSOURCES

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

For testing and logging the application uses jUnit and log4j.

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:

- Make it possible to configure db-relations in the Schema Mapping File, and handle these in the mappings.
- Intelligent error handling for OSM data.
- Further refactoring of the osm2gis.
- Adding new consistency checks.
- Write back.
    - Write via WFS and GeoServer into our database.
    - Find the new entries and add them over the OSM API to their database.
- Write an UI for configuring the Schema Mapping File.

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

0 DOCUMENTINFORMATION

0.1 CHANGE HISTORY

Datum	*Version*	*Änderung*	*Autor*
16.12.2009	0.1	Document created	ameier
17.12.2009	0.2	Adding or edit chapters (für Details siehe Dokument-History)	ameier
18.12.2009	0.3	Persönlicher Bericht hinzugefügt	jzimmerm
18.12.2009	0.4	CD Inhalt hinzugefügt	ameier

0 INHALT

1 AUFGABENSTELLUNG

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

1.1.1 EINFÜHRUNG

In einer vorangegangenen Bachelorarbeit mit dem Namen 'OpenStreetMap-in-a-Box' wurde eine Server-Applikation mit Datenbank, Geo-Webservices und einer Website erstellt. Im Wesentlichen dient OpenStreetMap-in-a-Box der Bereitstellung von Hintergrund-Karten für interaktive Kartenapplikationen im Web und auf Mobiles. Im Gegensatz zu bekannten Kartendiensten wie Google Maps bietet diese Lösung zwei entscheidende Vorteile: Erstens die freie Kontrolle über die Verfügbarkeit des Services (da 'in-house') und zweitens die Möglichkeit, eine an individuelle Bedürfnisse angepasste Kartengrafik zu definieren. Die Daten stammen vom OpenStreetMap-Projekt, dem 'Wikipedia' der Landkarten. Diese Karten, bzw. Geodaten sind oft detaillierter und aktueller als vergleichbare Produkte. Dies ist möglich durch die GPL-artige Lizenz der Daten. Die Software selber steht unter LGPL.

1.1.2 AUFGABENSTELLUNG

Das im Vorgängerprojekt entwickelte Kernstück, der Konverter (Java), der die bestehenden OSM-Daten in eine interne Datenstruktur importiert, soll überarbeitet werden, so dass die Konfiguration ein beliebiges Ziel-Datenschema erlaubt. Die dazu passenden Grafikkonfigurationen sollen wo möglich vereinfacht verwaltet werden können.

OpenStreetMap-in-a-Box soll also refaktorisiert und mit neuen SW-Komponenten (v.a. Präprozessor) ausgebaut werden und zwar wie folgt:

- Rewrite des gesamten Import-Konverters (osm2gis), so dass mit einem neuen Mapping (XML) ein beliebiges Ziel-Schema und Mapping (Tabellen/ Felder/ Werte) konfiguriert werden können (ev. Preprocessing-Schritt SQL-DDL generieren).
- Refactoring, v.a. das XML Parsing der OSM-Daten.
- Falls sinnvoll Konsistenzprüfung/-erhaltung des Datenschemas mit den GeoServer-Grafikkonfigurationen
- Integrieren der Konfiguration (GeoServer) von Teil B
- Integrieren der Website (-Erweiterungen) von Teil B (ev. an einen definitiven Platz).

Lieferdokumente (englisch wo angegeben, sonst deutsch):

- Installationsanleitung (englisch)
- Bedienungsanleitung
- Gesamter compilier-bereiter Sourcecode(englisch) inkl. Programmdokumentation sowie als Download gekennzeichnetes 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, wo sinnvoll mit Unit-Tests. Es wird Wert auf ausgetestete Software gelegt.
- Die Arbeit unterliegt einem separaten Non-Disclosure-Agreement. Diese liegt dem Dokument bei.

1.1.4 RANDBEDINGUNGEN, INFRASTRUKTUR, TERMINE UND BEURTEILUNG

- Randbedingungen Hardware/OS:
  - HW: keine Vorgaben
  - OS (für Webserver): Linux
- Software:
  - Java EE
- Termine und Beurteilung: gemäss Angaben auf www.hsr.ch. Beurteilung mit besonderem Augenmerk auf die Implementation.

Rapperswil, 8. Oktober 2009

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

2.1 PROBLEMBSTELLUNG

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

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

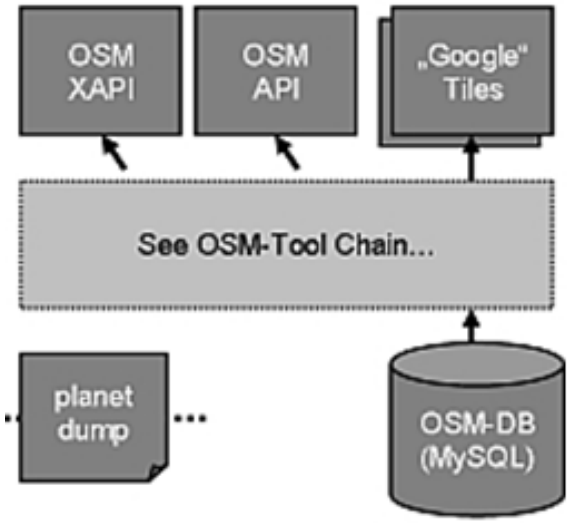


Abbildung 1: OSM-Tool-Chain

2.2 ZIEL DER ARBEIT

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

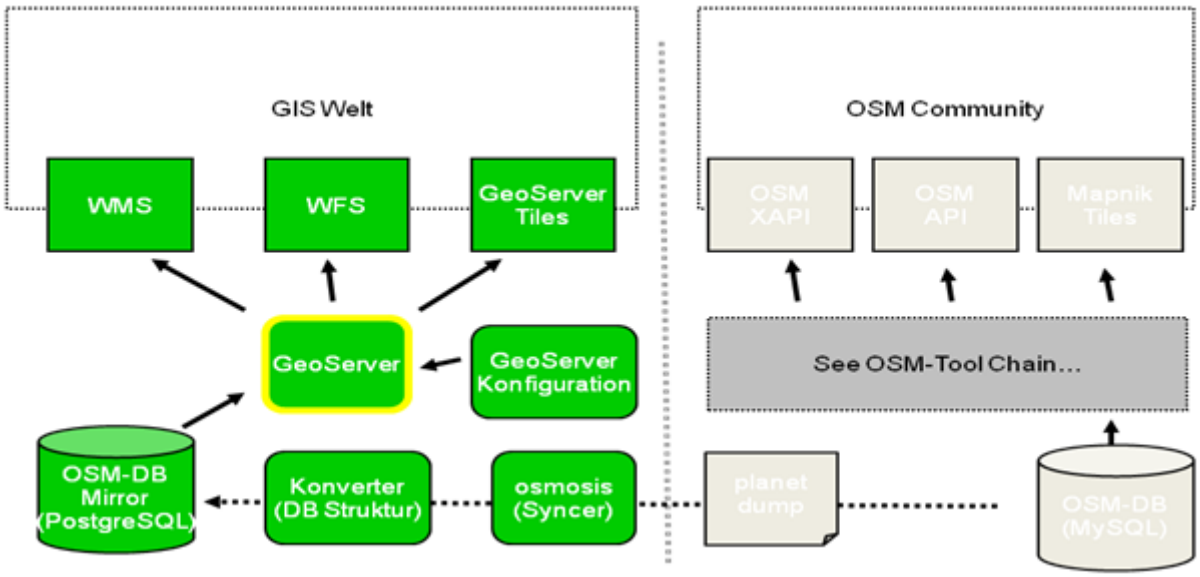


Abbildung 2: Ausgangslage (rechts) mit neuen Erweiterungen (links)

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

2.4 ERGEBNISSE

Bei Version 2.0 ist die Update Funktionalität nicht mehr unterstützt.

2.4.1 VOLLE KONFIGURATIONSMÖGLICHKEITEN

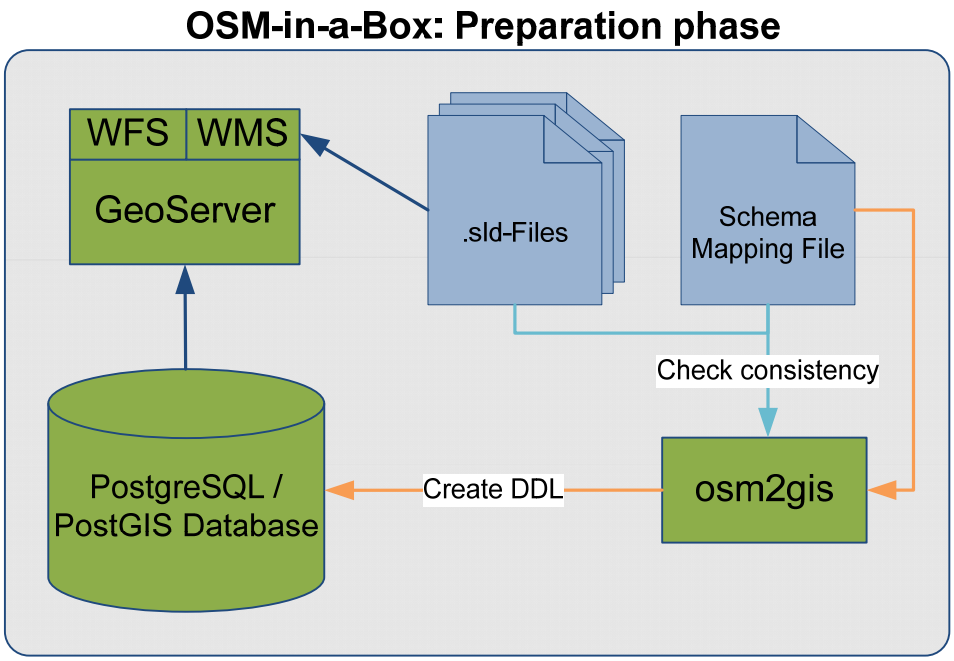


Abbildung 3: Vorbereitungsphase, Konfiguration

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.

2.4.2 EINBETTUNG DES OSM-IN-A-BOX SERVERS

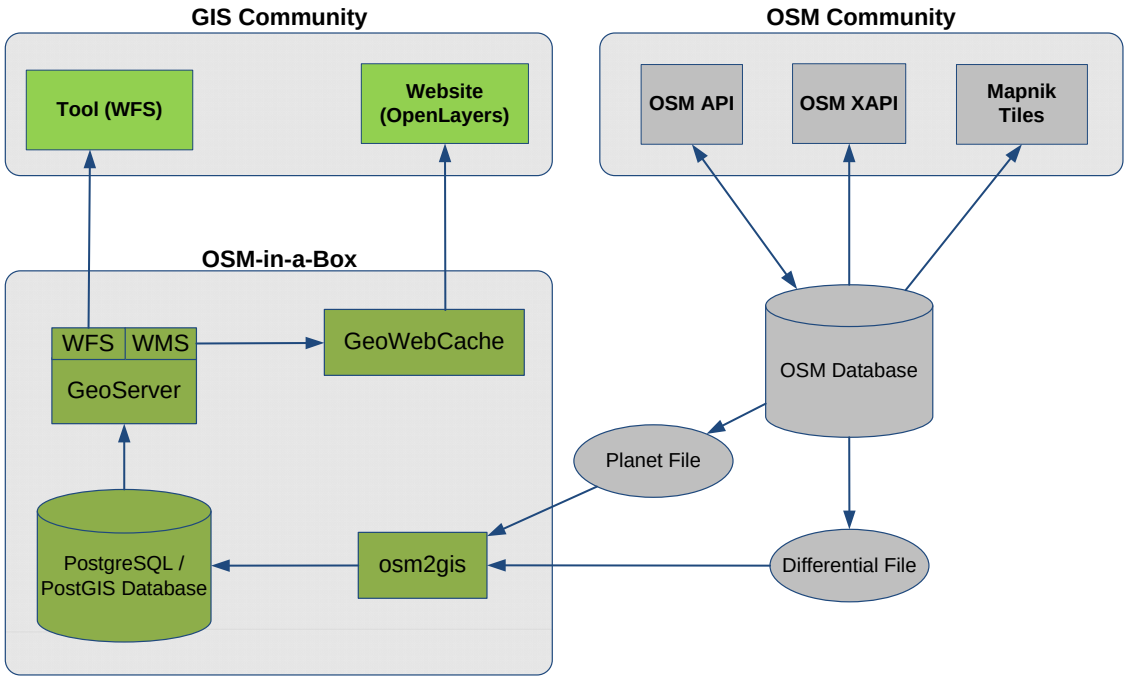


Abbildung 4: Ausgangslage (rechts) mit neuen Erweiterungen (links)

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 ist eine JavaScript Bibliothek welches es ermöglicht, auf einfache Weise eine

(OpenLayers)	dynamische Karte in eine Website zu integrieren.

2.4.3 OSM2GIS INITIALER IMPORT

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

Beispiel einer Kommandozeile für die Ausführung eines Initialen Imports:

osm2gis --initial-import -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.

2.4.4 OSM2GIS UPDATE SERVICE

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

Beispiel einer Kommandozeile welche ein regelmässiges Update startet:

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

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

2.4.5 OSM2GIS ARCHITEKTUR

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

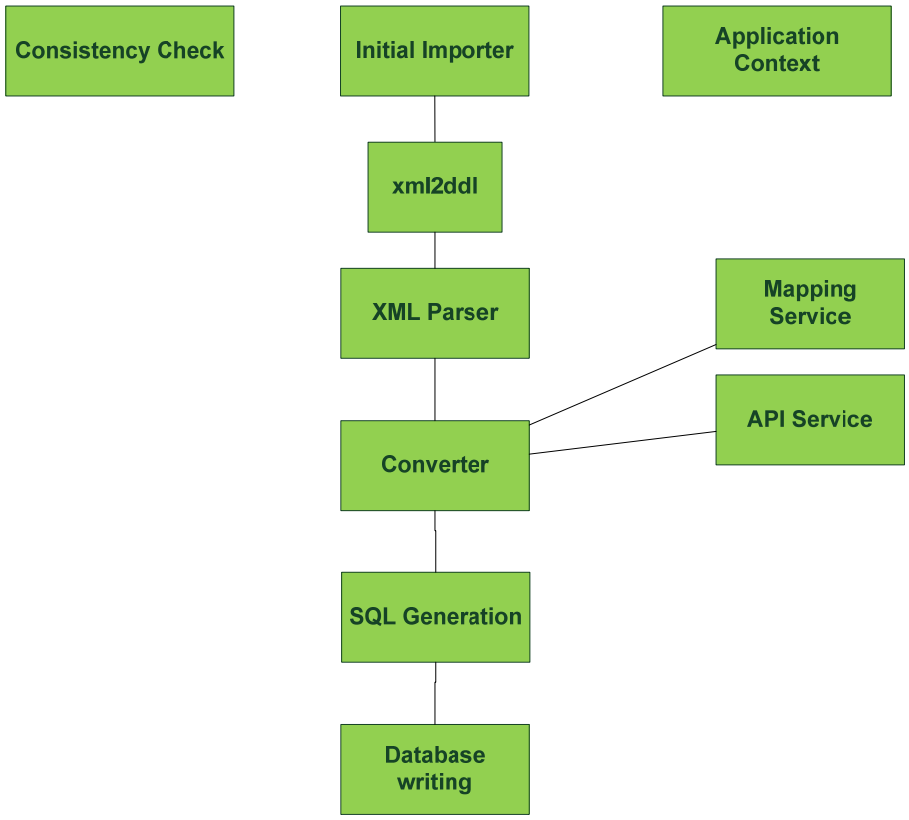


Abbildung 5: osm2gis Software Komponenten

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 SCHLUSSFOLGERUNGEN

Das Projekt konnte erfolgreich abgeschlossen werden. Die Unterstützung der Updates konnte aus Zeitgründen nicht implementiert werden. Die OSM-Daten können erfolgreich anhand der Regeln des erstellten Schema Mapping Files importiert werden. Auch kann auf das gesamte OpenStreetMap-in-a-Box System eine Konsistenzprüfung ausgeführt werden. Die Daten werden vom GeoServer dazu verwendet Kacheln zu generieren welche mittels WMS abgefragt werden. Die Vektordaten sind mittels WFS abrufbar.

2.6 AUSBLICK

Die osm2gis Applikation wurde so geändert und refaktorisiert das man ein eigenes Zielschema und die Regeln des Mappings definieren kann. Das Update muss in einem nächsten Schritt programmiert werden. Ausserdem sind zusätzliche Konsistenzprüfungen noch sinnvoll.

Das Fehlerhandling der OSM-Daten könnte zusätzlich noch erweitert werden sodass auch, bis zu einem gewissen Punkt, fehlerhafte OSM-Daten in die Datenbank importiert werden können.

Die Applikation könnte ausserdem so erweitert werden dass die Daten über das OSM API zurück geschrieben werden, was allerdings zu weiteren Problemen führen würde wie: Konsistenzhaltung der Daten. Es müsste eine Strategie entwickelt werden mit der veränderte Daten identifiziert werden können und die OSM IDs (unique identifiers für OSM Daten) korrekt gehandelt werden können.

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 detailliert 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 detaillierter 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.

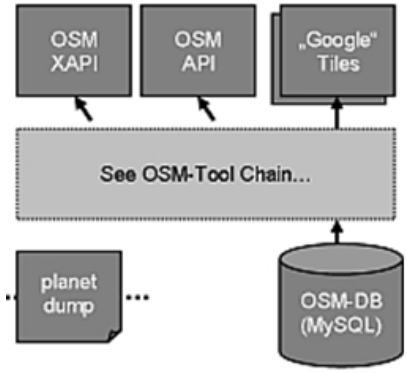


Abbildung 8: Ausgangslage

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.

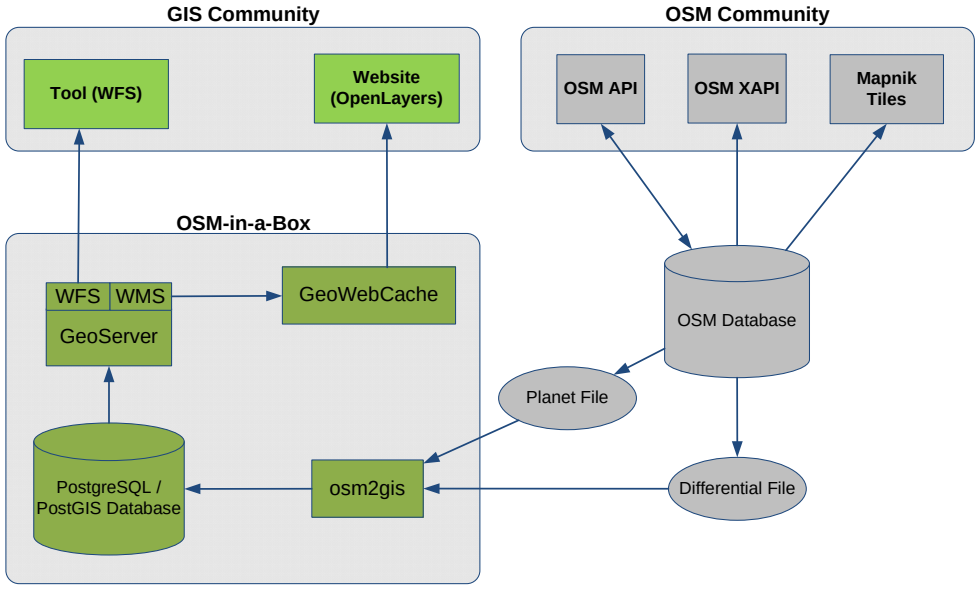


Abbildung 9: osm2gis Projektstruktur des Vorgängerprojektes.

3.1.2.3 ZIEL

Der osm2gis-Konverter soll in seiner Funktionalität erweitert und verbessert werden. Per Mapping-File sollen die zu importierenden OSM-Daten in ein beliebiges Zielschema konvertiert werden können. Die Angabe des Zielschemas muss vor dem erstmaligen Import der Daten geschehen, damit mittels Präprozessor die Datenbankstruktur erstellt werden kann. Einzelne Komponenten des Konverters müssen refaktorisiert werden (z.B. Parser), welche diese genau sein werden muss in der Analyse ermittelt werden.

Weiter soll eine Diff-Tool-Funktionalität eingebaut werden, welche die Mapping-Konfigurationen mit den Styles (Grafikkonfigurationen) des GeoServers vergleicht und Inkonsistenzen anzeigt. Somit wird die Konfigurationsarbeit des GeoServers vereinfacht.

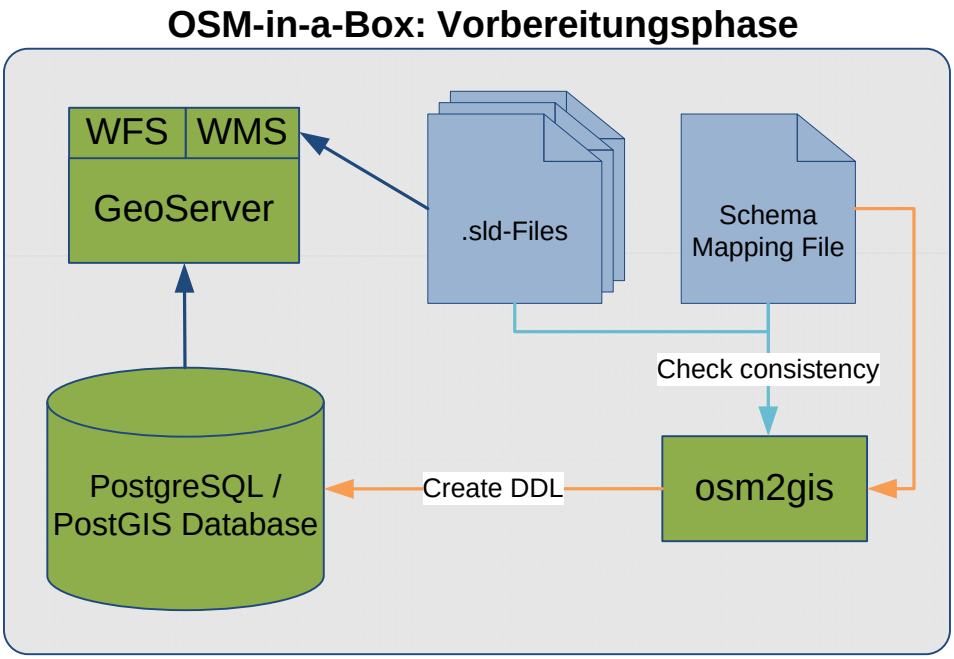


Abbildung 10:
Vorbereitungsphase in der einerseits anhand des Mapping-Files die Datenbankstruktur erzeugt wird und anderseits die .sld-Files (Grapfikkonfigurationsdateien) des GeoServers mit dem Mapping-File auf Konsistenz überprüft wird.

3.1.3 ANFORDERUNGEN IM DETAIL

Die Anforderungen an das Vorgängerprojekt sind im Dokument *Repository/root/tags/V0.2/Dokumentation/08_Endabgabe/Gesamtdokumentation.docx Kapitel 5.1* enthalten.

3.1.3.1 FUNKTIONALE ANFORDERUNGEN

Funktionalität	Beschreibung/Bemerkungen	Priorität	Abhängigkeit
Schemaerzeugung mit Mapping-File	Mittels Mapping-File soll die Datenbankstruktur dynamisch erzeugt (SQL DDL) werden können.	Hoch	
Konsistenzprüfung GeoServer / Mapping-File	Um die Konfigurationsarbeit beim GeoServer zu vereinfachen, sollen Inkonsistenzen zwischen der GeoServer Styles-Konfiguration und dem aktuellen Datenbankschema aufgezeigt werden können.	Mittel	Schemagenerierung mit Mapping-File

3.1.3.2 NICHT-FUNKTIONALE ANFORDERUNGEN

Lokale Datenbank

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

Mapping-File

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

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 gestalten.

GeoServer

- Durch die Erweiterungen in diesem Projekt darf der vorbereitete GeoServer in seiner Funktionalität nicht eingeschränkt werden.
- Mittels entsprechendem Mapping-File soll dieselbe Datenstruktur erzeugt werden können wie im Vorgängerprojekt, womit der GeoServer mit bestehender Konfiguration wie bisher funktionieren muss.

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

- Integration der Diplomarbeit von Marco Busarello
- Migration der Website an einen definitiven Platz

3.1.4 USE CASES

3.1.4.1 USE CASE DIAGRAMM

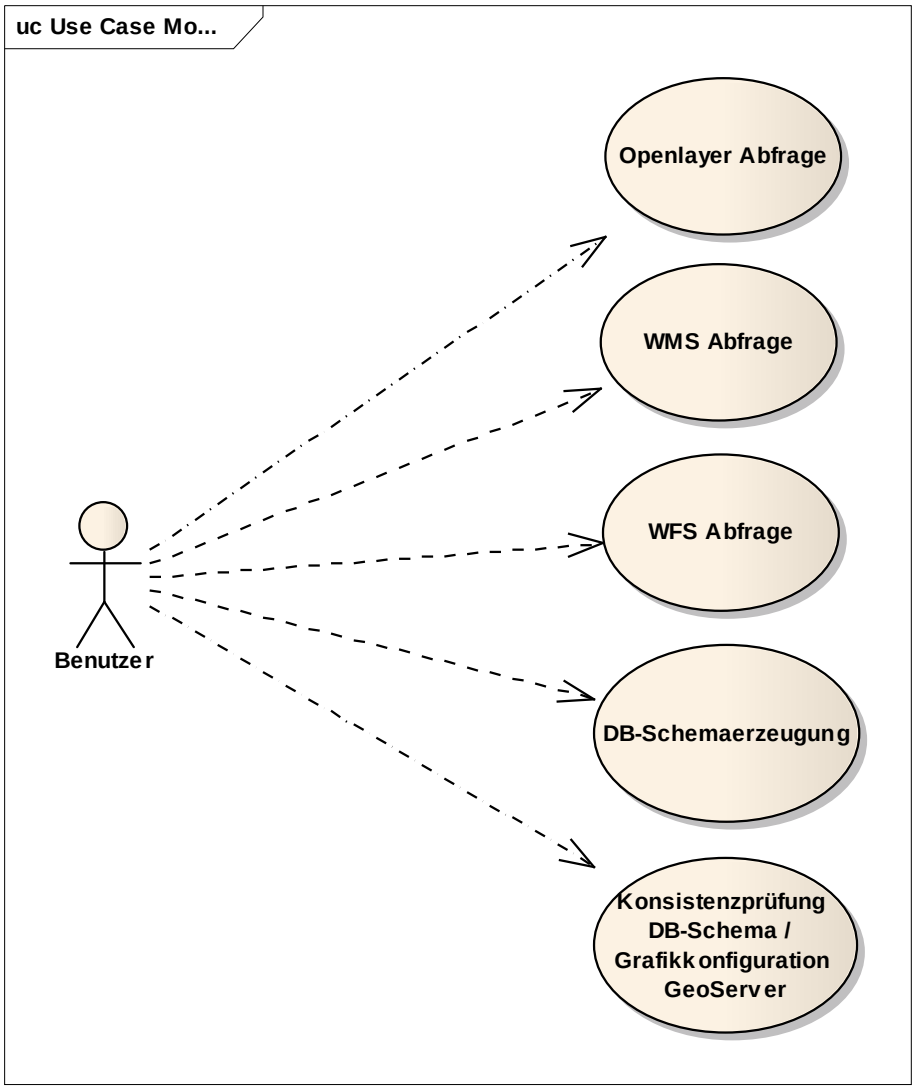


Abbildung 11: Use Case Diagramm mit den oberen drei Use Cases des Vorgängerprojektes und den zwei neuen Use Cases

3.1.4.2 USE CASES ÜBERSICHT

Als Benutzer ist der Server- und Websiteadministrator gemeint, welcher

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

3.1.4.3 USE CASES DETAILIERT

3.1.4.3.1 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	Der Endbenutzer ruft die Seite mit dem Anfahrtsplan auf. 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.

3.1.4.3.2 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 (Eingebettet 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

3.1.4.3.3 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.

3.1.4.3.4 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.

3.1.4.3.5 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.

3.1.4.3.6 UC6 INITIAL UPDATE

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.

3.1.4.3.7 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.

3.1.4.3.8 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 Zielschemas in der Datenbank sobald er einen Init Import ausführt. Die in den Mappingkonfigurationen konfigurierten Daten sind auf der Karte ersichtlich.
STANDARDABLAUF	1. Änderung des Config Files 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.

3.2 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 build in the prior bachelor thesis and updated to the current version 2.0 in this semester 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 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 2.0 NOTES

One main goal to achieve in version 2.0 of the osm2gis application was the user-defined configuration of the database schema and the mapping rules which map osm-entities to specified tables. Due to the complexity of this task and time-shortage, the applications update mechanism for fetching differential osm planet files couldn't be updated to support the customized configuration. This task will be done in the next version. Nevertheless, this document explains the update process from the previous version even if it's disabled in the current state.

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
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 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.

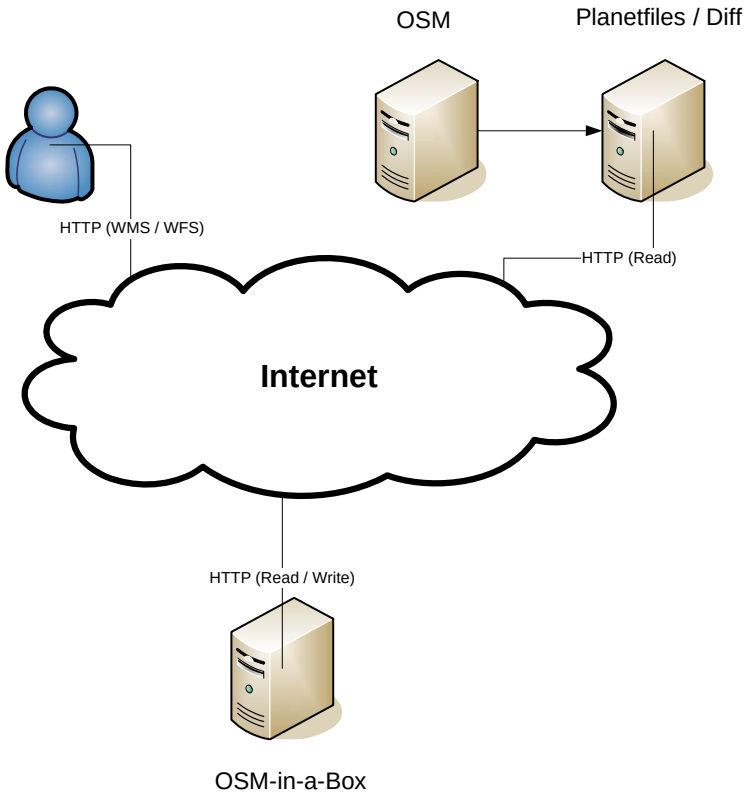


Figure 3: Physical architecture

3.2.3 LOGICAL ARCHITECTURE

3.2.3.1 OVERVIEW

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

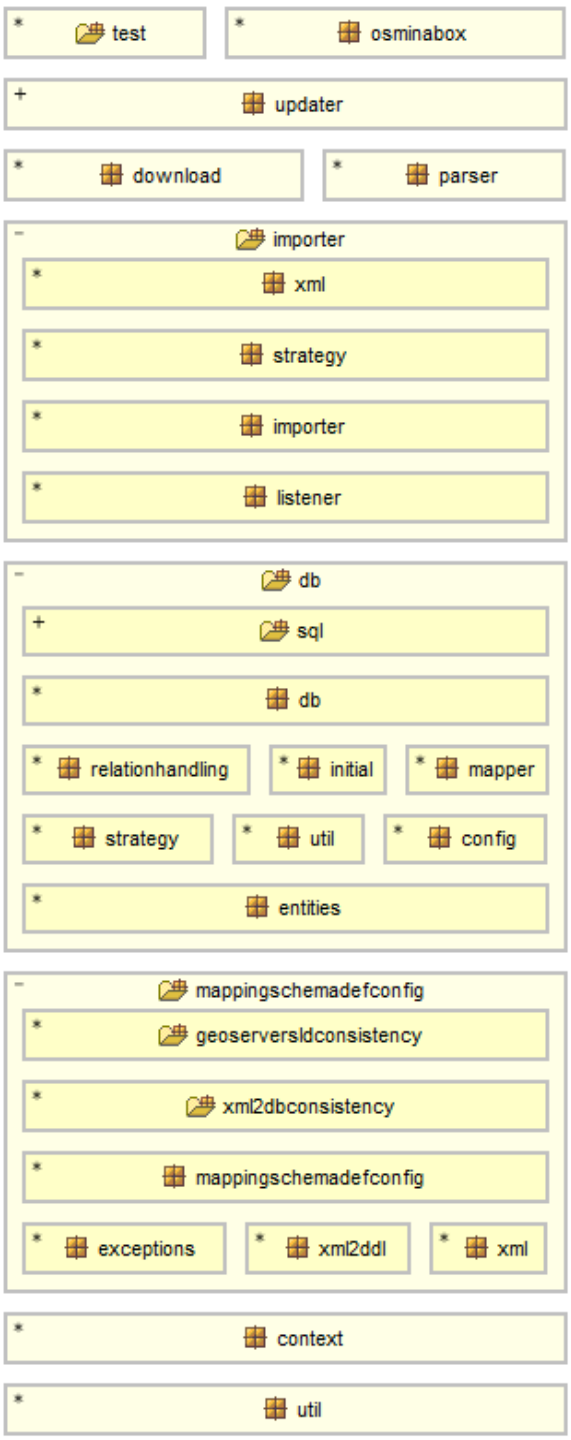


Figure 4: Rough overview of the package structure

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

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

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

The *db* packages manage the creation of the SQL-Statements for an initial import and the differential updates. The db services are usually called from the package importer.

The *mappingshemadefconfig* 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 (Mapping Shema 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.

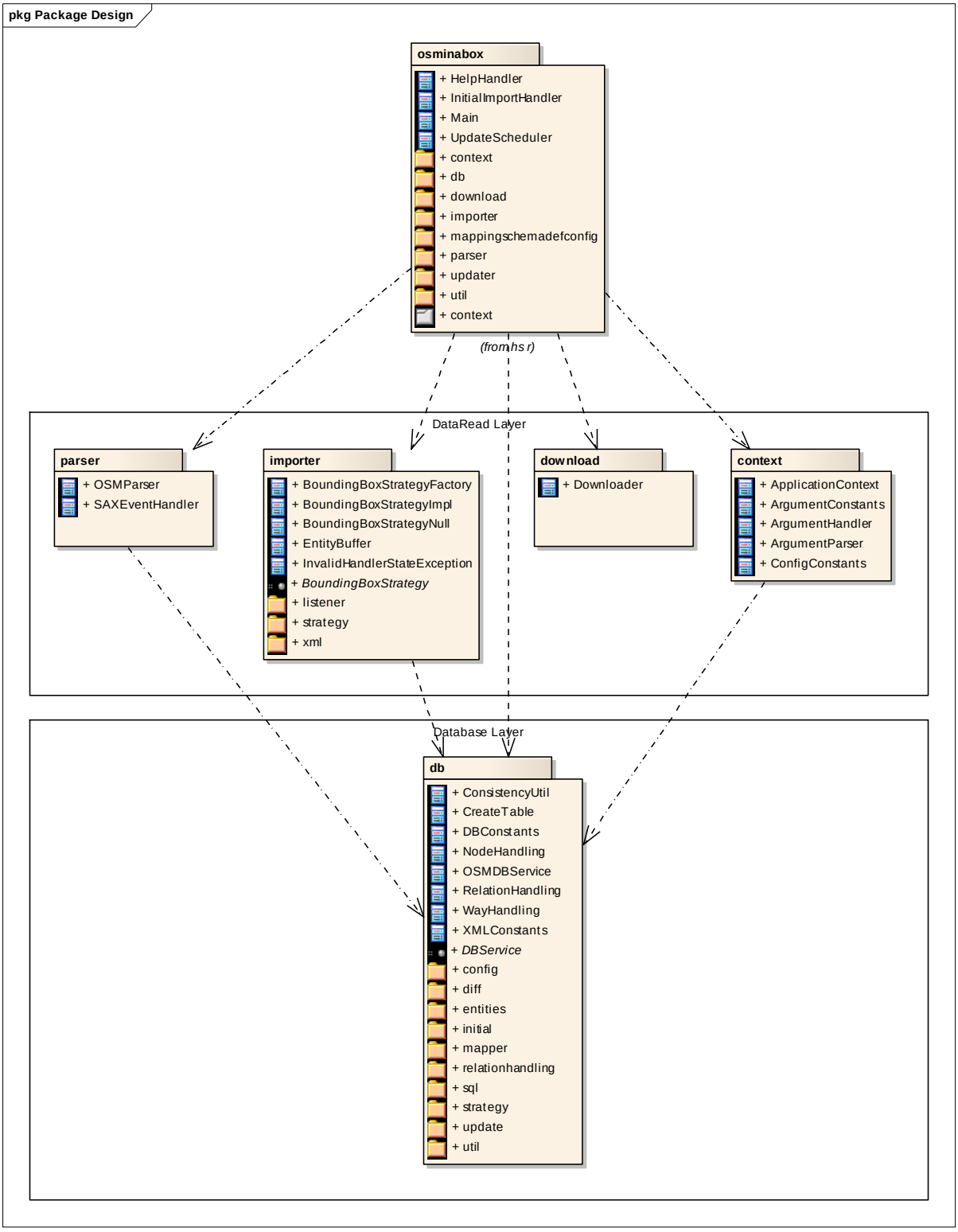


Figure 5: Package dependencies

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

Not supported in version 2.0

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 and 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:

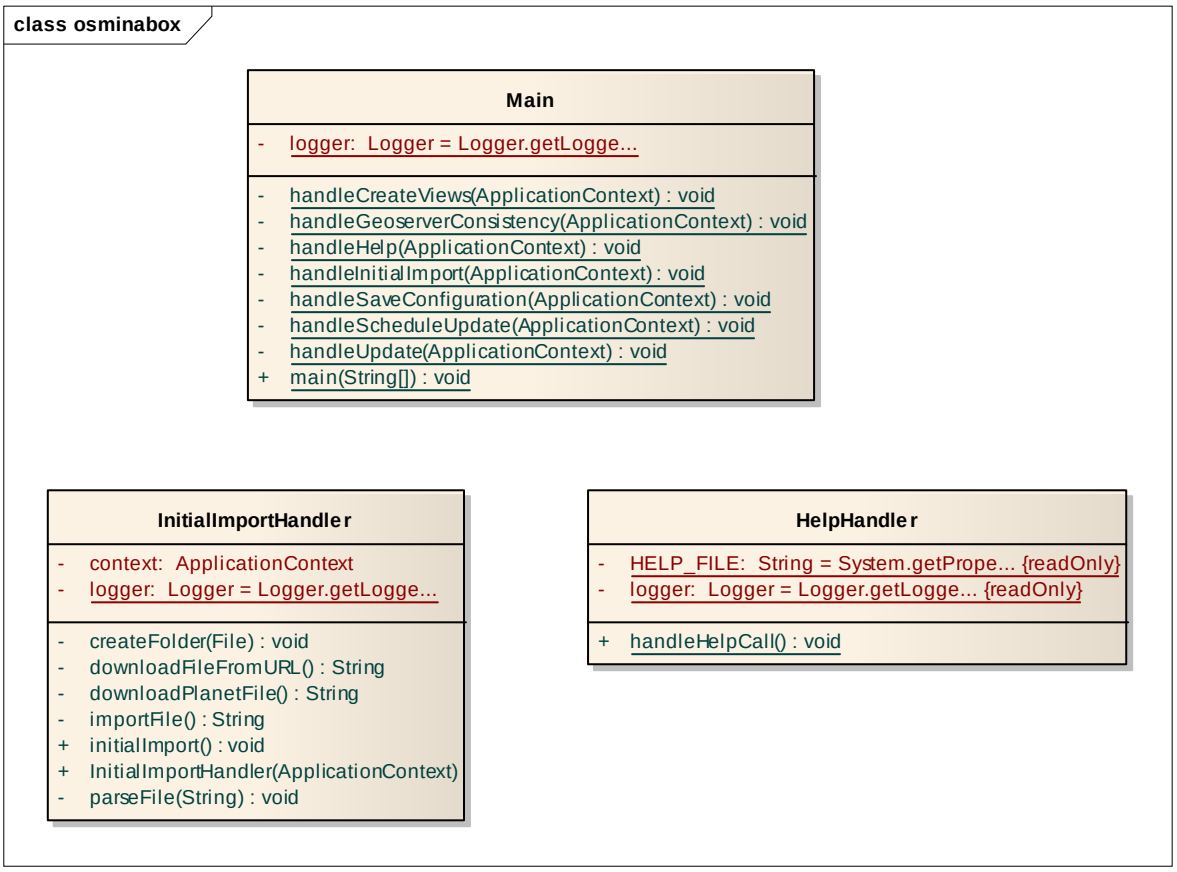


Figure 6: Application initialization

INITIALIZATION SEQUENCE

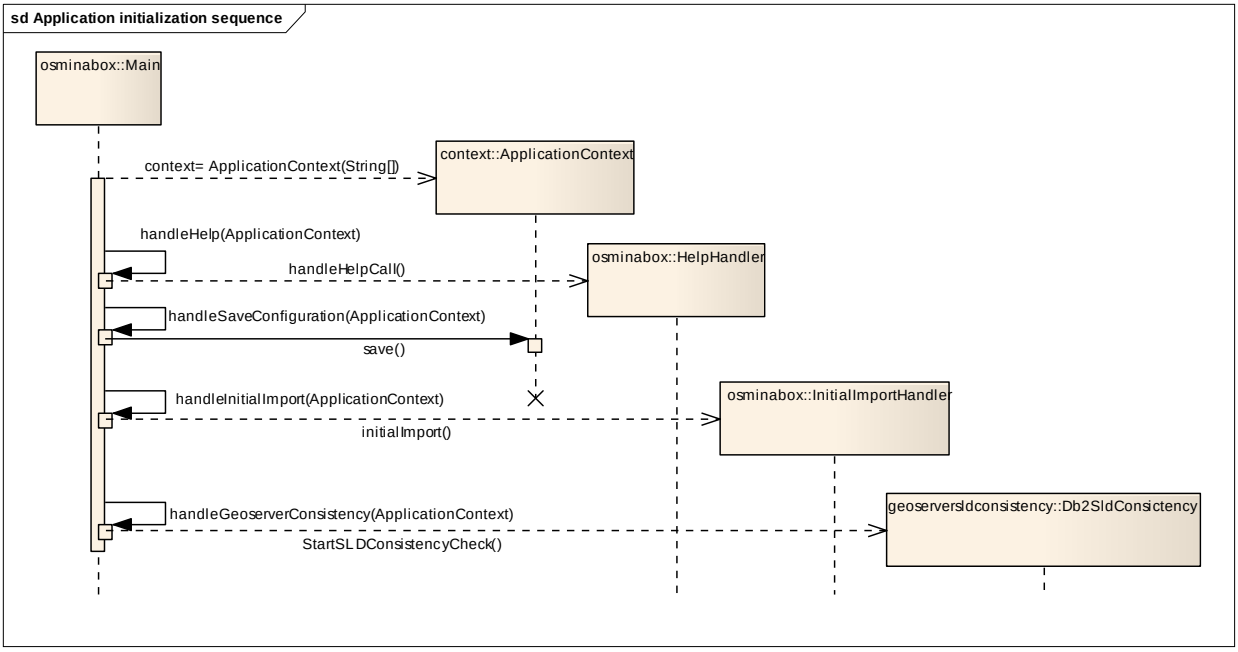


Figure 7: Initialization Sequence

Explanations

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

- InitialImportHandler
  Handle the initial import of OSM-Planet files or shape files. A shape file is basically a subset of a planet file.
- HelpHandler
  Prints the 'man' page to the standard output.
- Db2SldConsitency
  Generates the report about the consistency of all configuration files (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

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

INVOLVED PACKAGES

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

IMPLEMENTATION APPLICATIONCONTEXT

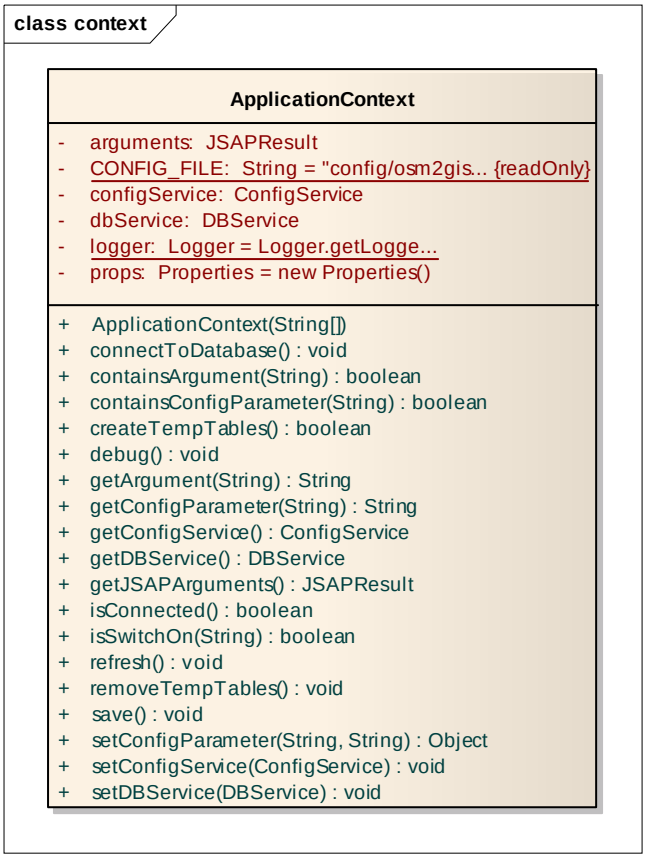


Figure 8: ApplicationContext

Explanation

The ApplicationContext provides methods to get and set configuration parameters. 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. But the only thing the relations are important for, within the application, is to connect multiple ways to one area. Real relations like a train route which references multiple railway-ways, can't be handled at this point and need to be implemented in further versions.

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 ways. So there is a need to store the nodes in order to process the ways and so on. With the streaming parser technology there is no 'Look back' or no 'Look ahead' at the moment an xml element occurrence is received.

The solution

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

3.2.3.5.2 FUNCTIONALITY PARSING PROCESS

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

3.2.3.5.3 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.parser	Contains the main parser classes
ch.hsr.osminabox.importer.	Classes for importer and bounding box strategy
ch.hsr.osminabox.importer.xml	XML Handlers
ch.hsr.osminabox.importer.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

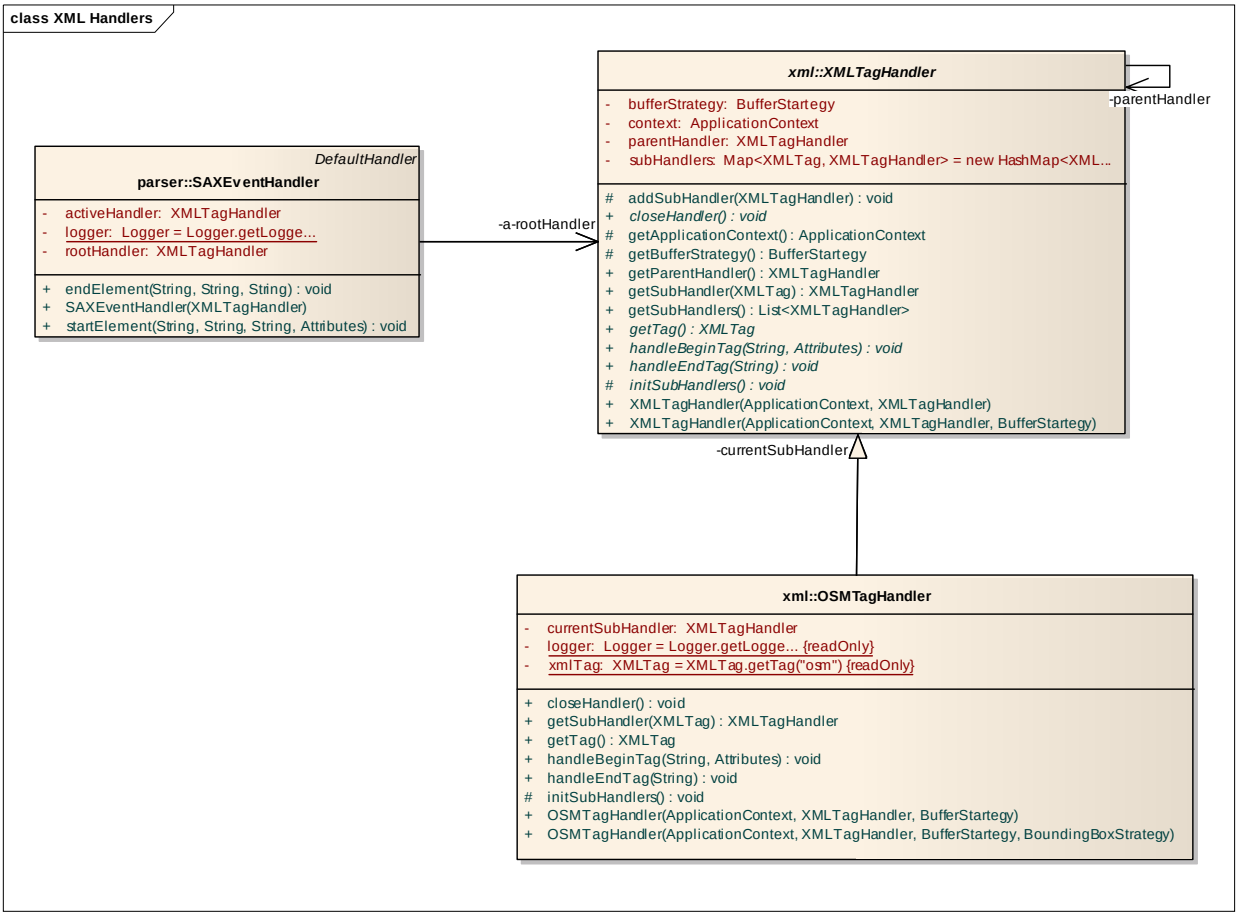


Figure 9: Root handler

Explanation

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

3.2.3.5.4.2 TAG HANDLER IMPLEMENTATIONS

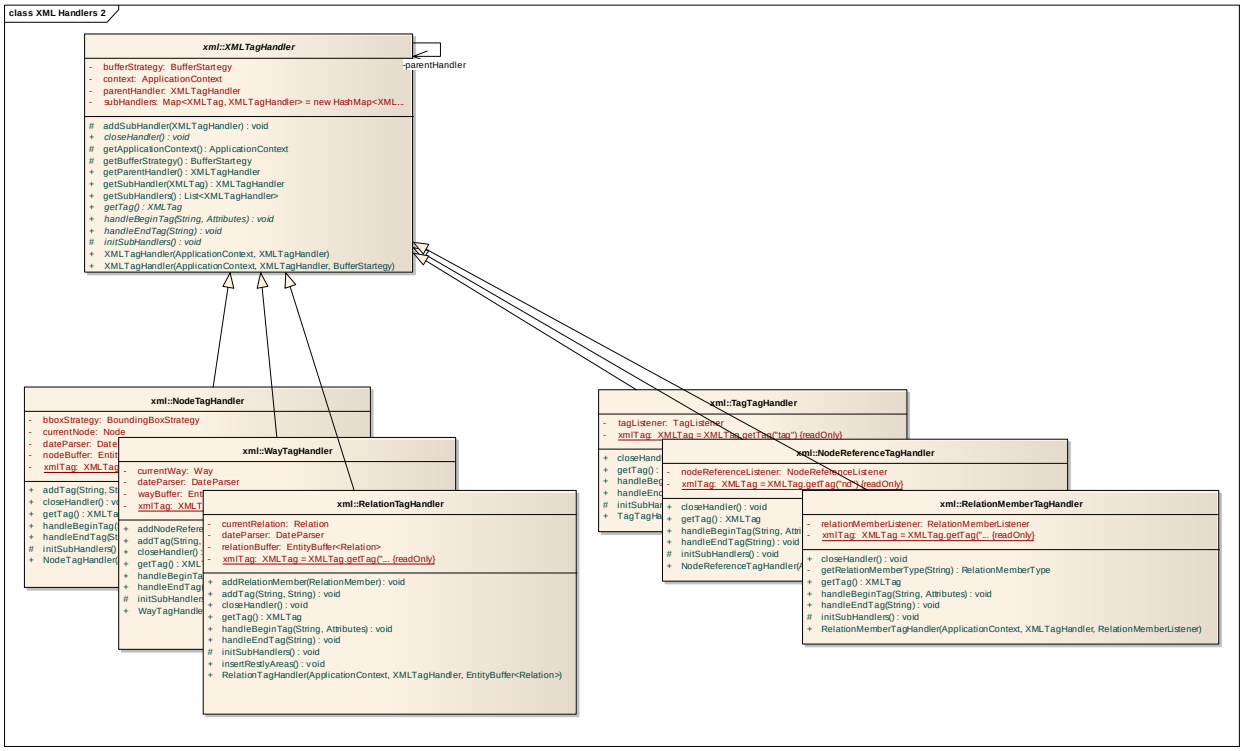


Figure 10: Tag Handler

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 way or a node within a relation. Can be a sub element of 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.

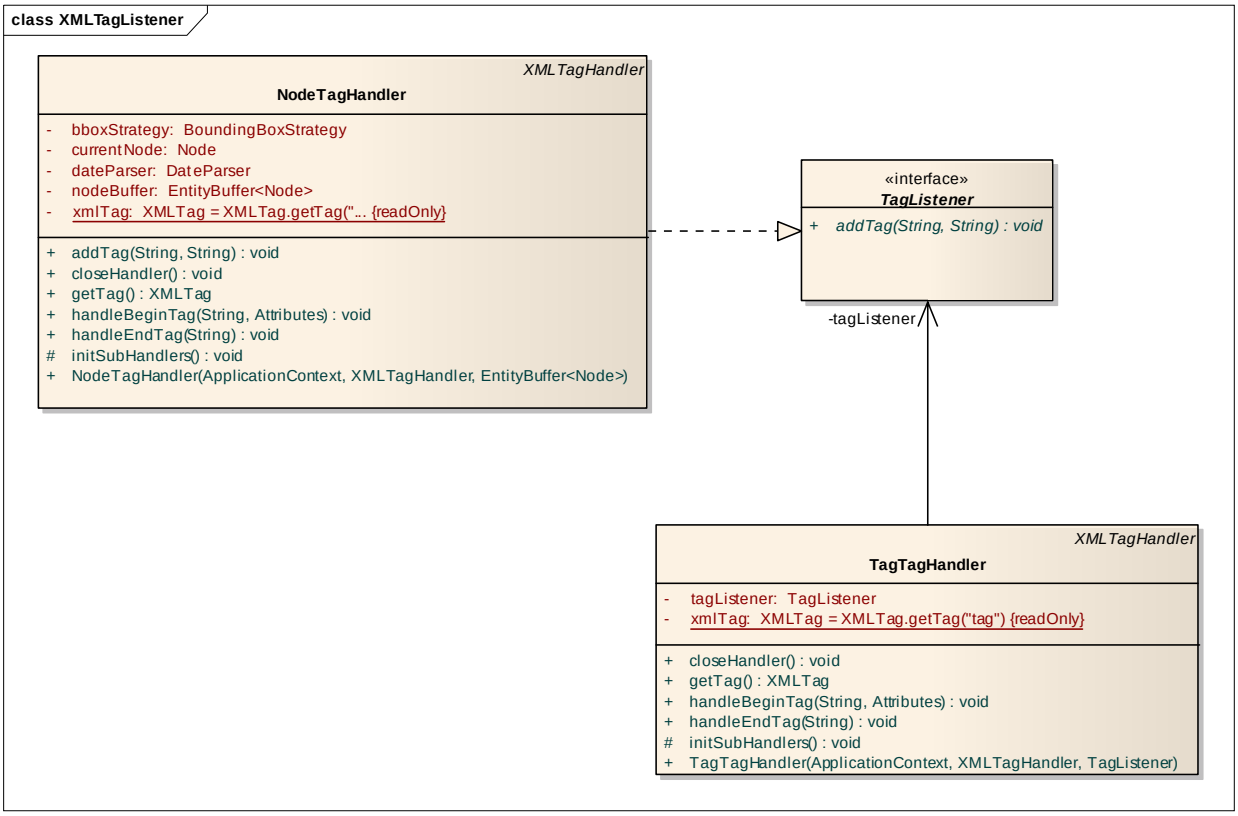


Figure 11: Handling Subtags

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.

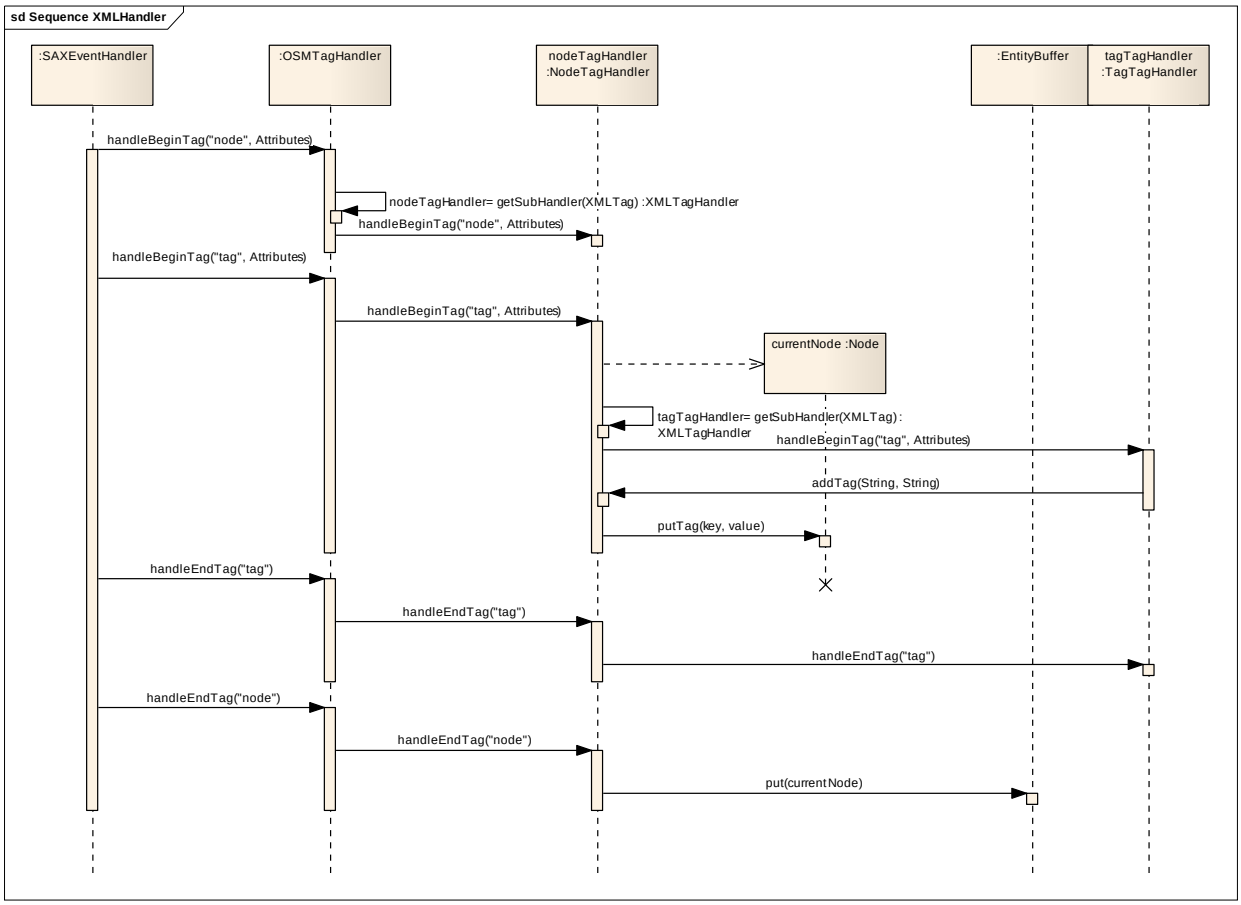


Figure 12: Tag Handling

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. This chapter describes what happens to the entities that are created along this process.

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.

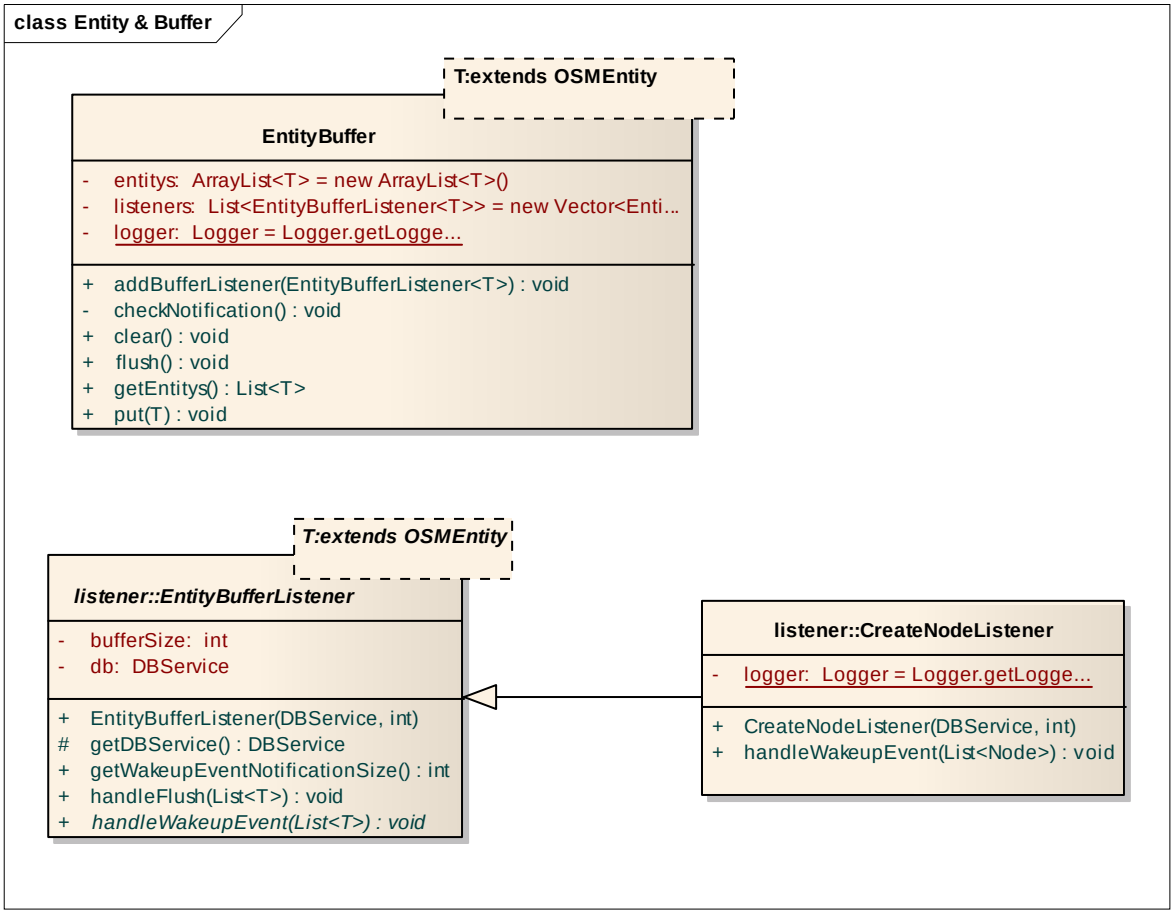


Figure 13: Buffer Strategy

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 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.

3.2.3.5.7 BOUNDINGBOX STRATEGY

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

In version 2.0 of osm2gis, all nodes in the parsed file need to be stored temporarily even if they don't pass the boundingbox-test since they might be referenced by ways. This is to eliminate the problem that ways (or relations) won't be inserted in the database because some of their referenced nodes / ways are missing.

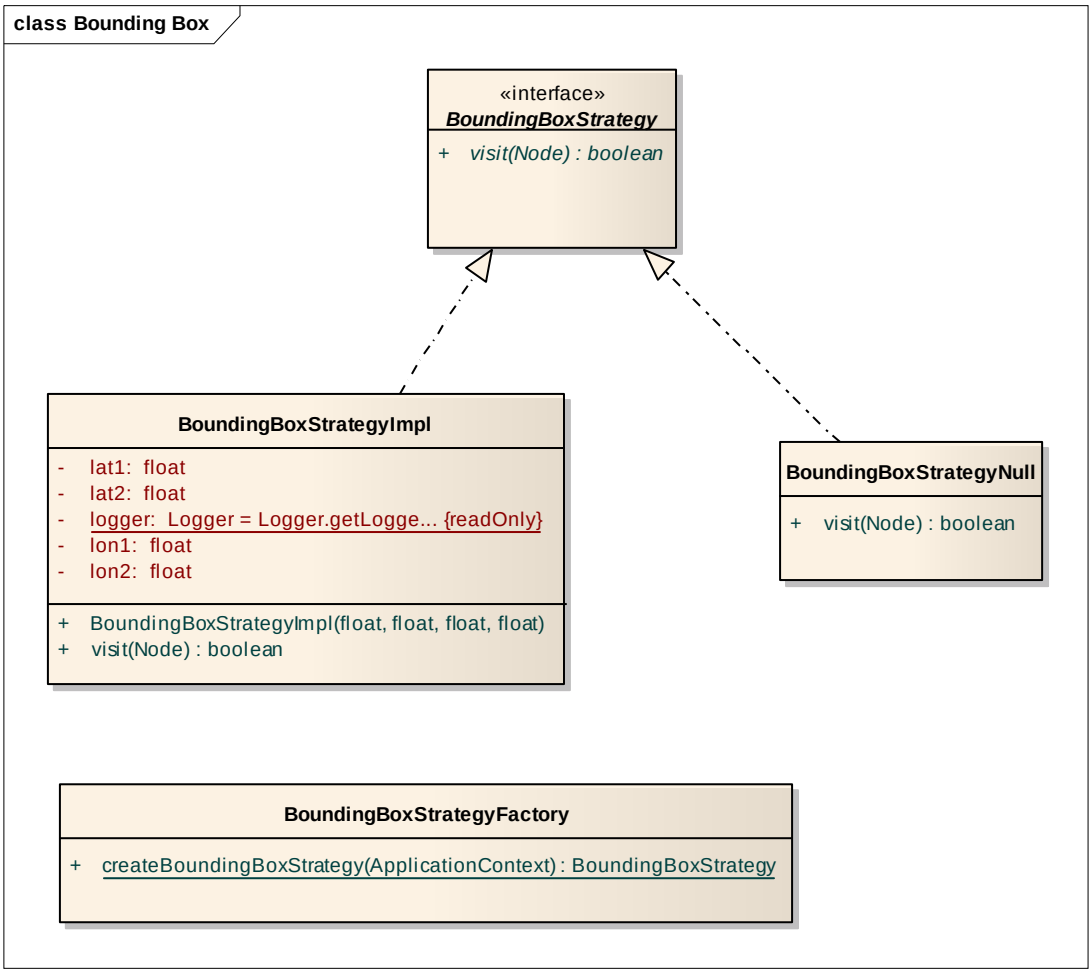


Figure 14: Boundingbox Strategy

Explanation

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

3.2.3.6 DATABASE: DATA MIGRATION

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

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

NORMAL PROCEDURE OF DATA IMPORT/ DIFFERENTIAL UPDATE

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

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

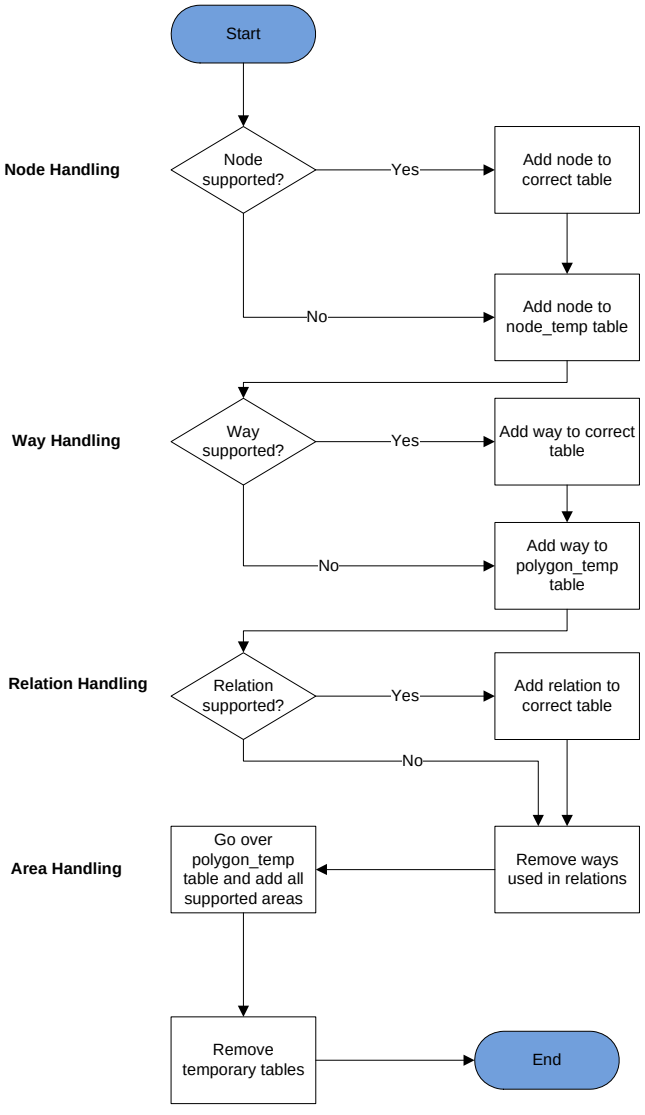


Figure 15: Data Import/Differential Update

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.

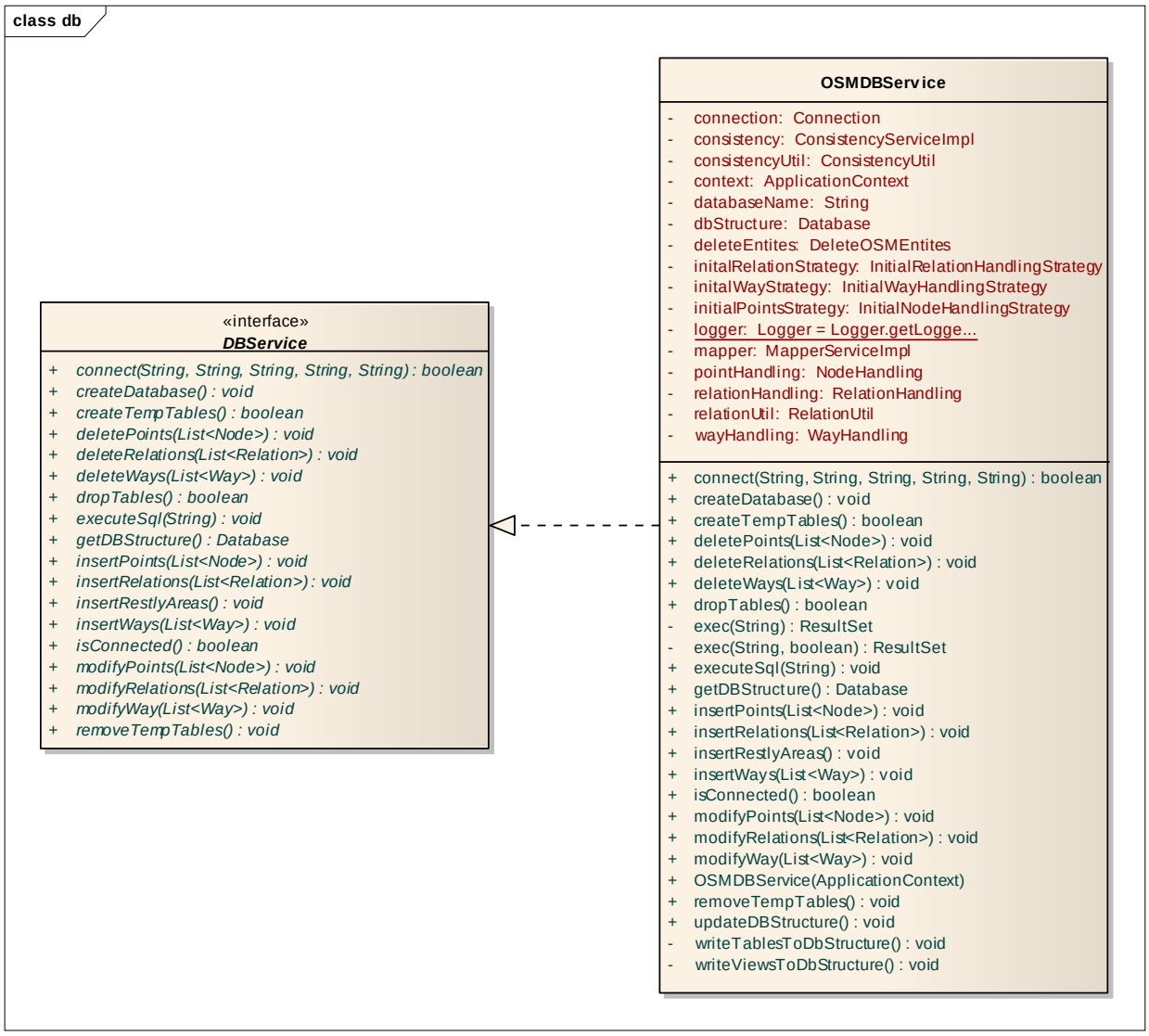


Figure 16: Database Layer

3.2.3.7.1 STRATEGY PATTERN

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

Problem:

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

Solution

- The solution was to add a 'global' strategy pattern to the application. The three OSMEntity-Types (node, way and relation) have their own interface. Two implementations of every OSMEntiys-Type are created. One for initial import and one for differential update. In the end the 'if, else' code, which decides the Database Table, appears only once in the whole application.
- The Strategies are initiated in the OSMDBService and used if needed.

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

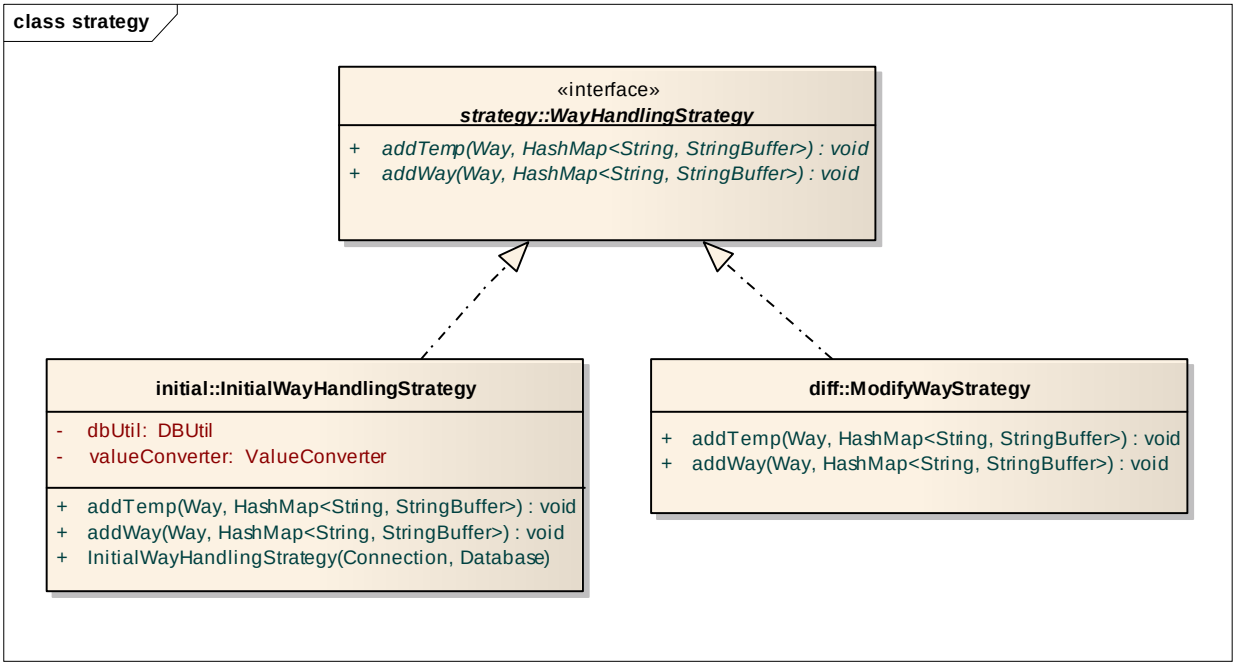


Figure 17: Strategy Pattern

3.2.3.8 NODE HANDLING

3.2.3.8.1 FUNCTIONALITY

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

- Creation of SQL statements for the different database tables on initial and on differential import. Tables are completle user-defined but all need a column “geom” with the datatype “geometry” to be used by the GeoServer.

3.2.3.8.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method insertPoint(..) and modifyPoint(..)
ch.hsr.osminabox.db.strategy	Node handling strategy
ch.hsr.osminabox.db.initial	Start of the generation of the sql scripts
ch.hsr.osminabox.db.sql	Creation of the initial node values
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.diff	Creation of the differential values
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization and value conversion

3.2.3.8.3 IMPLEMENTATION INITIAL NODE HANDLING

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

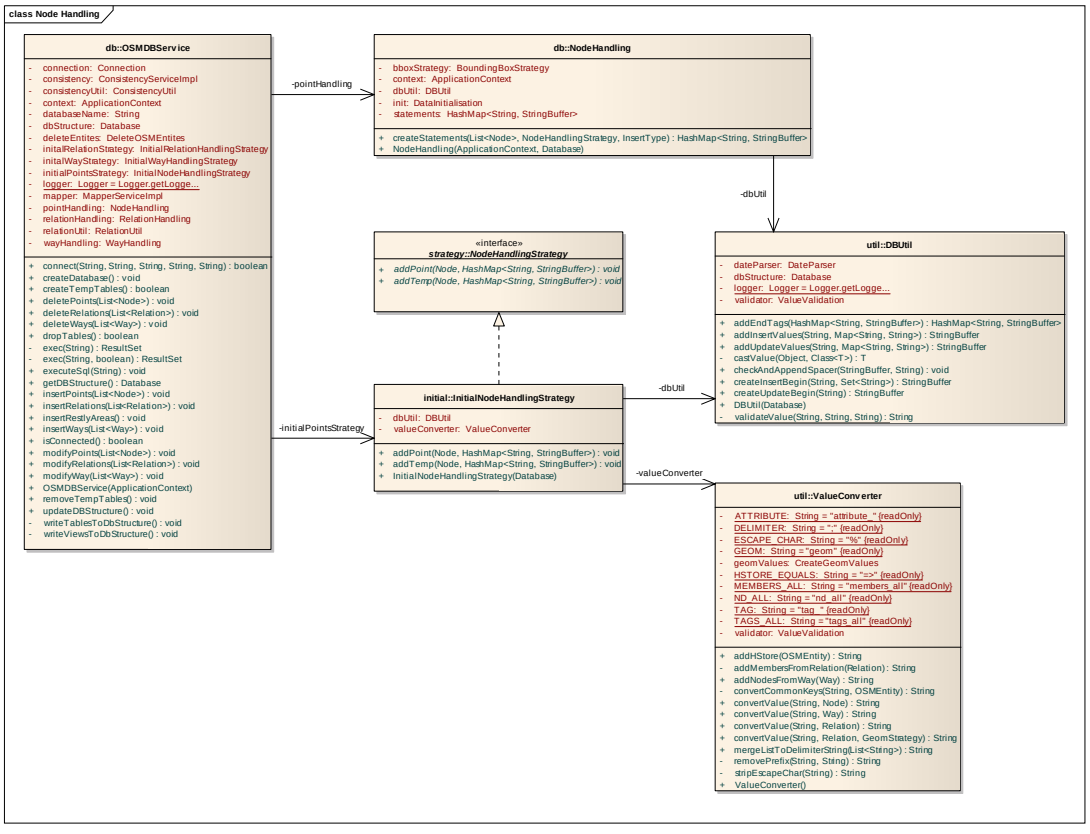


Figure 18: Implementation Initial Node Handling

3.2.3.8.3.1 NODEHANDLING CLASS

Package: ch.hsr.osminabox.db.NodeHandling

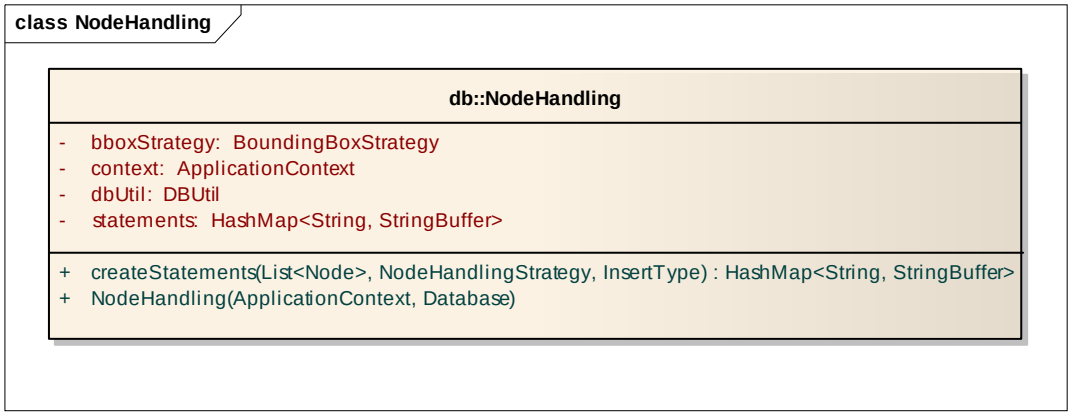


Figure 19: NodeHandling Class

This is the starting class of the NodeHandling. Both (initial and update) use this class, but they pass their own NodeHandling strategy to the createStatement(..) method.

The returned HashMap contains <String tablename, StringBuffer SQL statement>. There is one SQL statement generated for all given nodes (on initial import).

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

3.2.3.8.3.2 SEQUENCE DIAGRAMM INITIAL

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

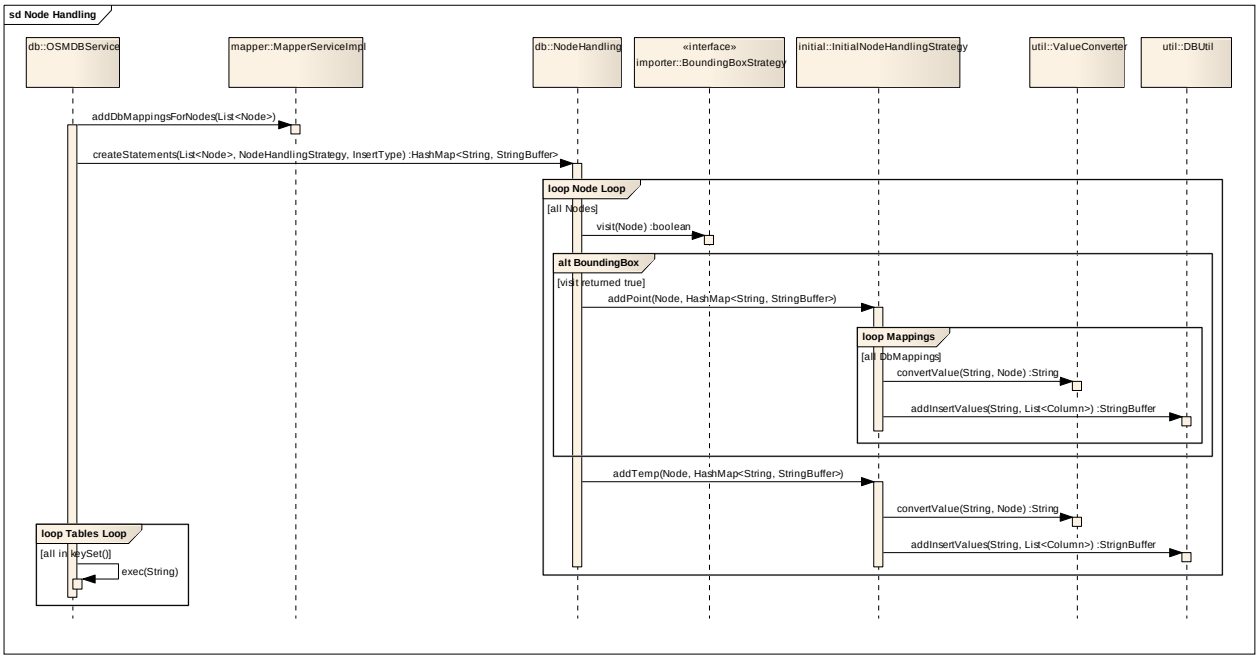


Figure 20: SD Initial Node Handling

Explanations

1. *importer calls the method insertPoints(List<Node>).
    a. OSMDBService calls addDBMappingsForNodes onMapperServiceImpl which adds the mapping information from the Schema Mapping File to all nodes in the list. See also the according chapter.
    b. Afterwards the method createStatements on NodeHandling is called. As HandlingStrategy we pass the InitialNodeHandlingStategy, because an initial import has to be handled. InsertTyp is: INSERT
2. NodeHandling
    a. The NodeHandling class loops over all nodes in the list and checks if the Node lies within the BoundingBox by calling the visit-method on the BoundingBoxStrategy.
    b. If it is, the Node is passed to the addPoint-method of the InitialNodeHandlingStrategy.
3. InitialNodeHandlingStrategy
    a. Initializes the SQL statement (if it is not already done by a previous call)
    b. Loops over all DbMappings found on this Node (A Node can be inserted in multiple tables if it matches their criteria).
    c. Loops over all Columns defined in the specific DbMapping.
    d. Converts the value from the DbMapping into the “real” value (see MapperService Chapter) by calling the convertValue-method for each Column.
    e. Adds the converted value to the sql String by passing the Column which contains the table name and all values to the DBUtil class.
4. Since every Node needs to be inserted into the node_temp-table, its Columns are created and the same mechanism as before is used to create the according sql.
5. NodeHandling
    a. After adding all nodes to their according StringBuffers in the HashMap, a semicolon will be added at the end of every statement.
6. OSMDBService now loops over the values of the resulting HashMap and executes every SQL Statement.

3.2.3.9 WAY HANDLING

3.2.3.9.1 FUNCTIONALITY

The core functionality is to create SQL Insert 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.

3.2.3.9.2 INVOLVED PACKAGES

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

3.2.4 IMPLEMENTATION INITIAL IMPORT

Classes that are used for an Initial Import on Ways are shown below.

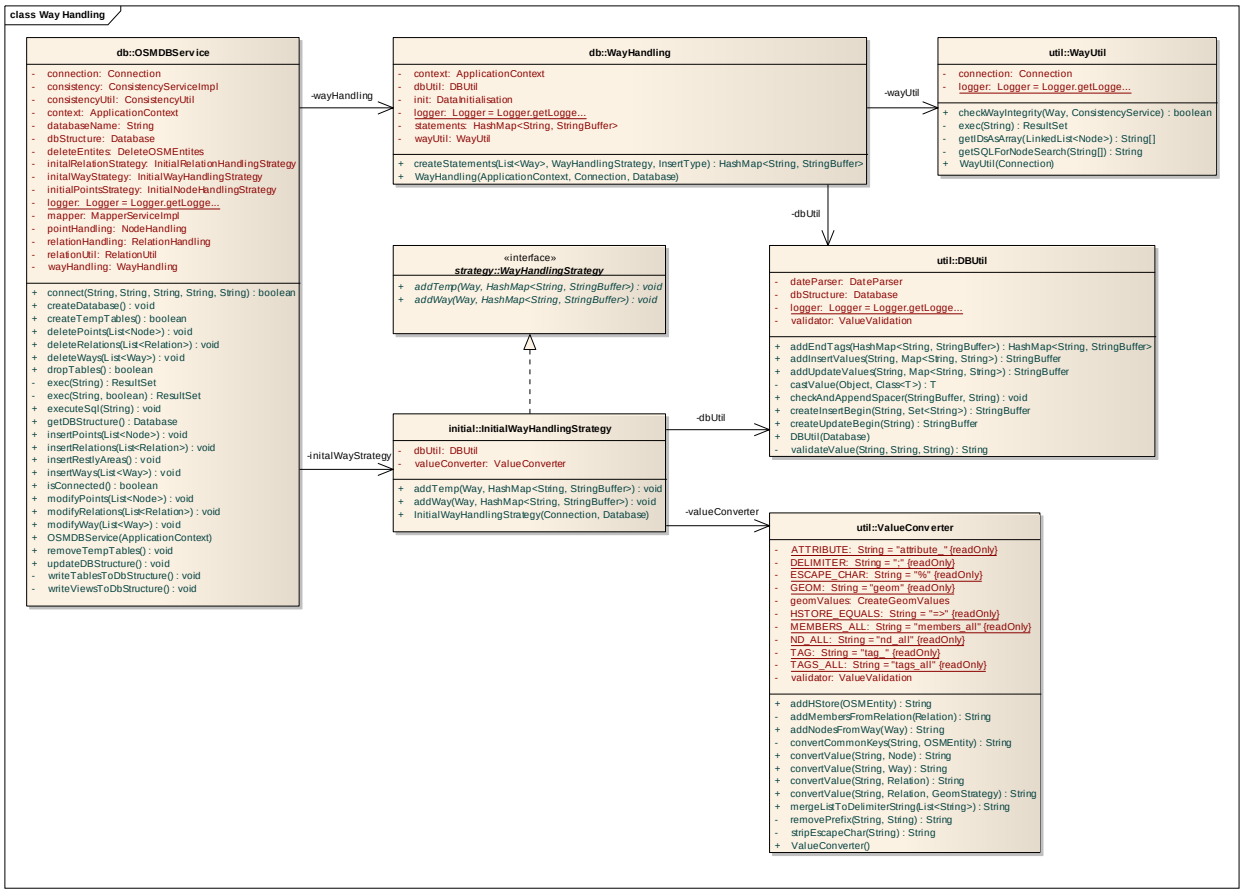


Figure 21: Initial Way Handling

3.2.4.1.1.1 SEQUENCE DIAGRAM

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

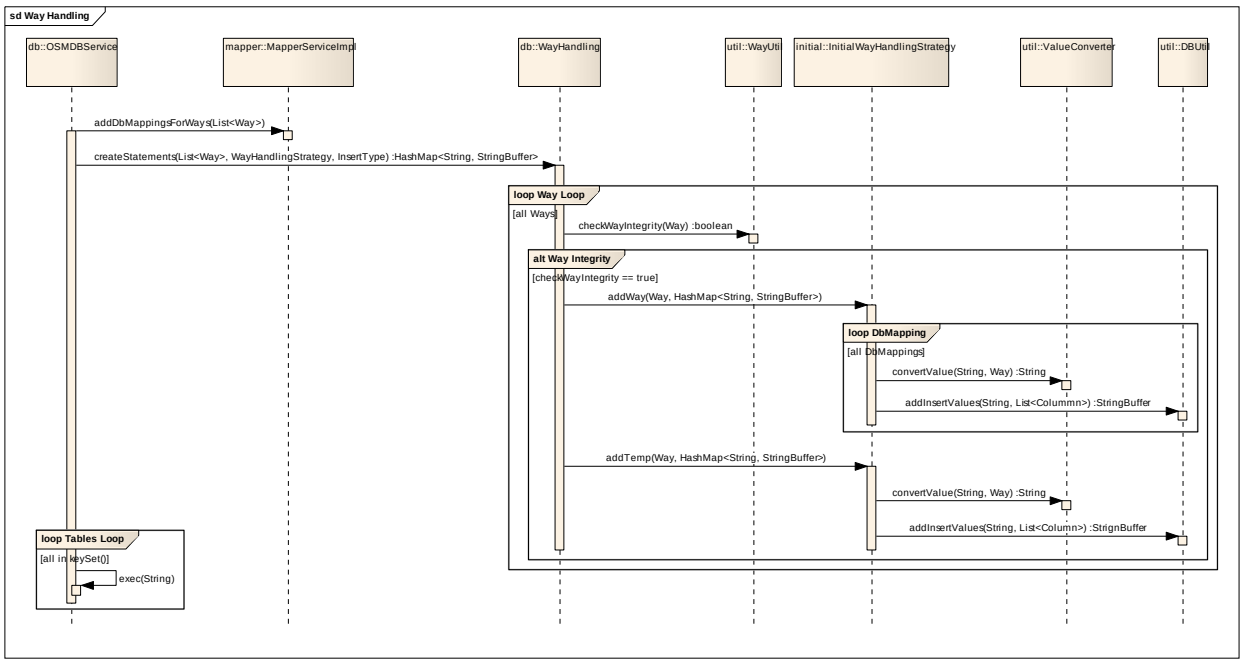


Figure 22: SD Initial Way Handling

Explanations

1) *.importer calls on OSMDBService the method insertWays(List<Way>)
   a) OSMDBService calls addDBMappingsForWays onMapperServiceImpl which adds the mapping information from the Schema Mapping File to all ways in the list. See also the according chapter.
   b) Afterwards the method createStatements on WayHandling is called. As HandlingStrategy we pass the InitialWayHandlingStategy, because an initial import has to be handled. InsertTyp is: INSERT
2) WayHandling
   a) The WayHandling class loops over all ways in the list and makes an Integrity-check (are all referenced Nodes inavailable in the node_temp table?) by calling the checkWayIntegrity-method on the WayUtil class.
   b) If all referenced Nodes from a Way are available, the addWay-method of the InitialWayHandlingStrategy class is called.
3) InitialWayHandlingStrategy
   a) Initializes the SQL statement (if it is not already done by a previous call)
   b) Loops over all DbMappings found on this Way (A Way can be inserted in multiple tables if it matches their criteria).
   c) Loops over all Columns defined in the specific DbMapping.
   d) Converts the value from the DbMapping into the “real” value (see MapperService Chapter) by calling the convertValue-method for each Column.
   e) Adds the converted value to the sql String by passing the Column which contains the table name and all values to the DBUtil class.
4) Since every Way needs to be inserted into the polygon_temp-table, its Columns are created and the same mechanism as before is used to create the according sql.
5) WayHandling

a) After adding all ways to their according StringBuffers in the HashMap, a semicolon will be added at the end of every statement.

6) OSMDBService now loops over the values of the resulting HashMap and executes every SQL Statement.

3.2.4.2 RELATION HANDLING

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

3.2.4.2.1 FUNCTIONALITY

Here a list of the functionality implemented on relation handling

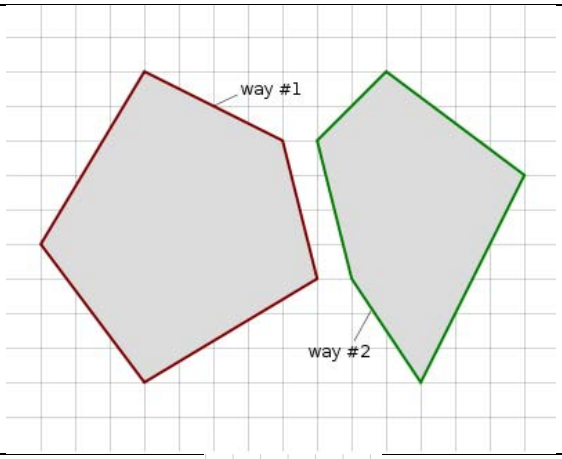
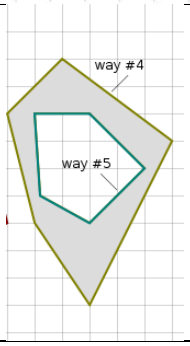
- Creation of SQL Statements for initial imports and differential updates
- Detection of relation type
- Special handling of relation area's with inner and outer ways.
- Detection and removal of combined ways

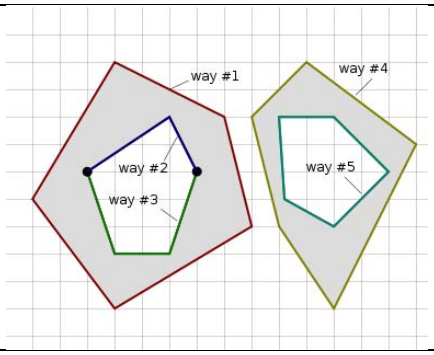
3.2.4.2.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method insertRelation(..) and modifyRelation(..)
ch.hsr.osminabox.db.strategy	Relation handling strategy
ch.hsr.osminabox.db.relationhandling	RelationTypeDetection
ch.hsr.osminabox.db.sql	Creation of the initial relation values
ch.hsr.osminabox.db.sql.util	Util package for creating sql statements
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization and RelationUtil

3.2.4.2.3 RELATIONTYPES

Given of the relation data type from OpenStreetMap we have to support three Types of Relations

1.	The first Type is *only outer*. This means that the relation 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.	

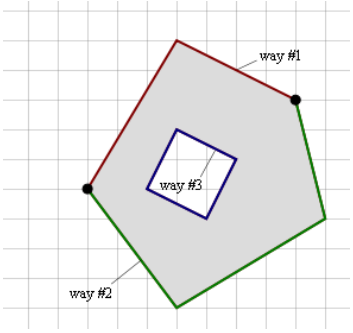
3.2.4.2.4 COMBINED WAYS

OpenStreetMap implemented some specialties. One of them is that a boundary of a polygon can be formed by more than 1 way. What was thought of as a simplification, turned out to be much more complicated the interpreting application has to search itself for this combined ways. There is nothing tagged inside the relation that would make it clear that these ways are combined.

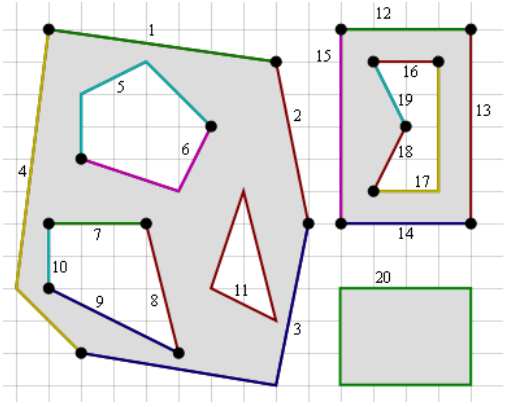
For this feature we had to implement an algorithm that removes those combined ways and consolidates them into one big way because PostGIS is not able to handle boundaries of area's which are not one line.

Here we have some examples of Combined Relations. One you have already seen in the chapter before.

An easy exmaple would be this:



This of course can be done much more Complicate.



3.2.4.2.5 IMPLEMENTATION WAY COMBINING

Here you see a short view of the implementation of this algorithm

1. We start at the top of the way list and take the last node-ID of the first way
2. Then take the first node-ID of the next way.
3. Compare these 2 IDs
    a. If they are equal AND the nodes are not from the same way, we combine them and set the validity of one way to false.
4. If they are not equal, we start again, but take as first way the way we just had as second and so on.
5. We do this until no more ways have to be combined.

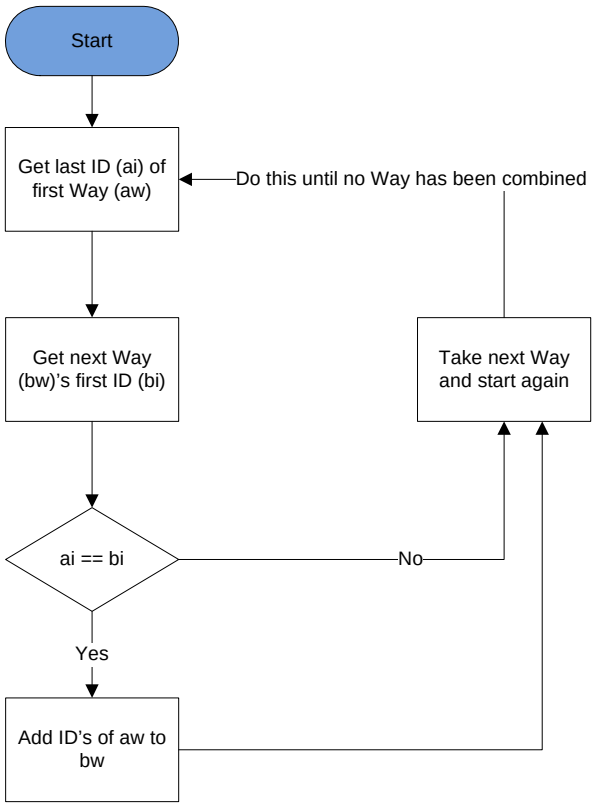


Figure 23: Way combining

3.2.4.2.6 IMPLEMENTATION RELATION TYPES

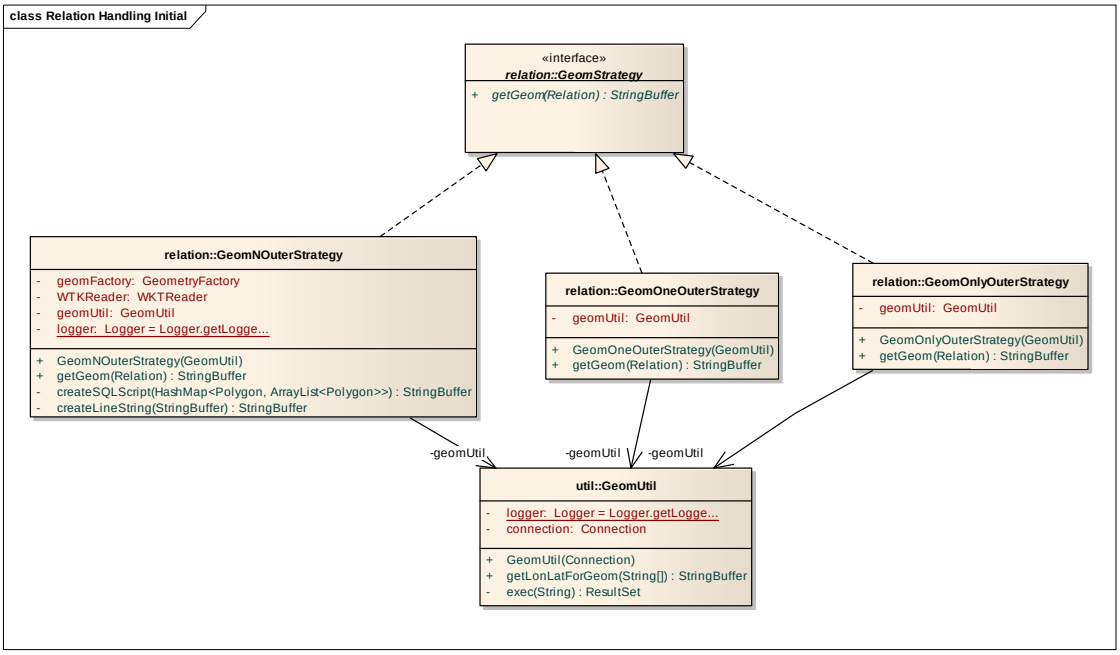


Figure 24: Relation types

The relation types do only have effects on the geom. Creation. We implemented a standard GeomStrategy interface which all classes that create a geom have to implement.

3.2.4.2.7 IMPLEMENTATION INITIAL IMPORT

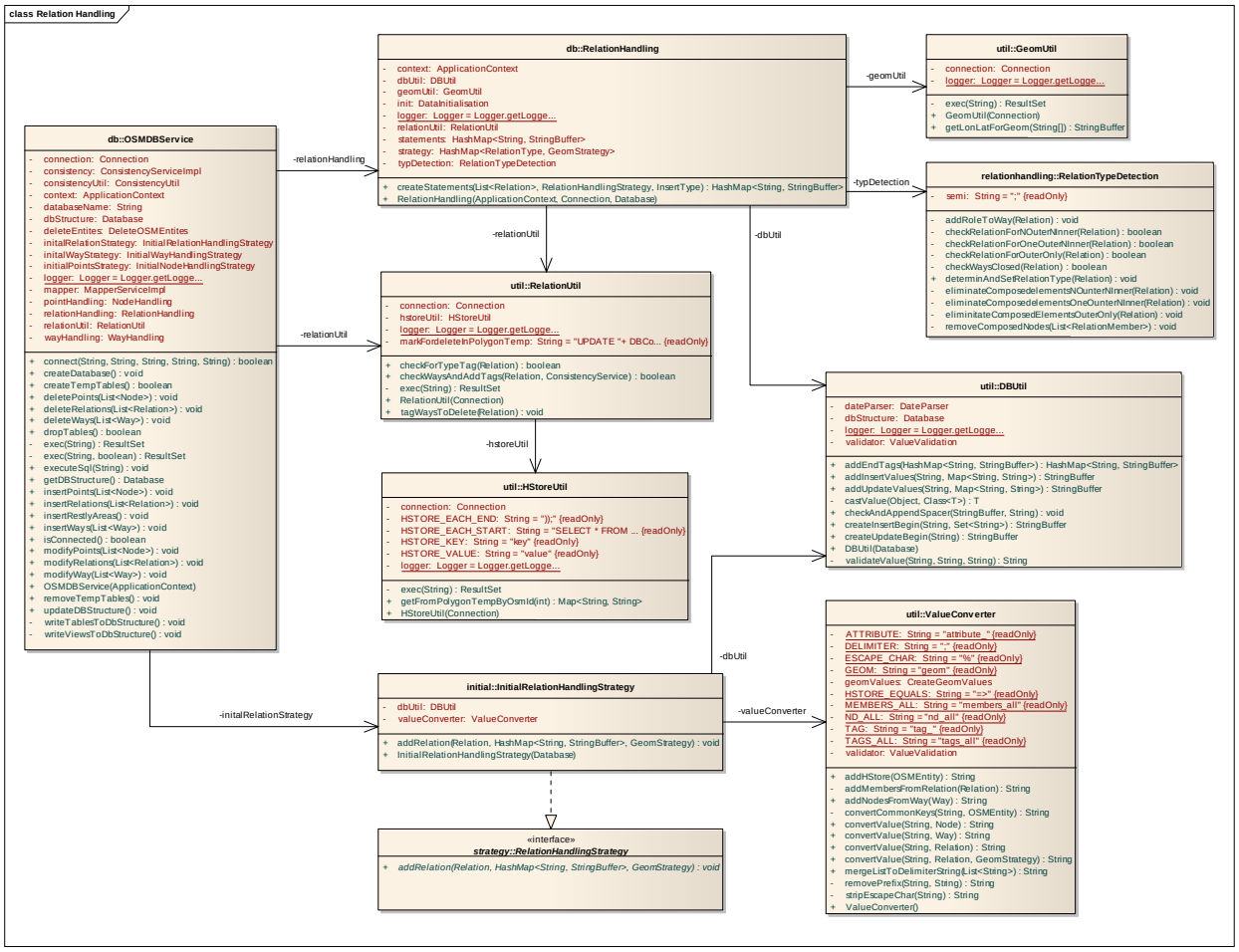


Figure 25: Initial Relation Handling

3.2.4.2.7.1 SEQUENCE DIAGRAM

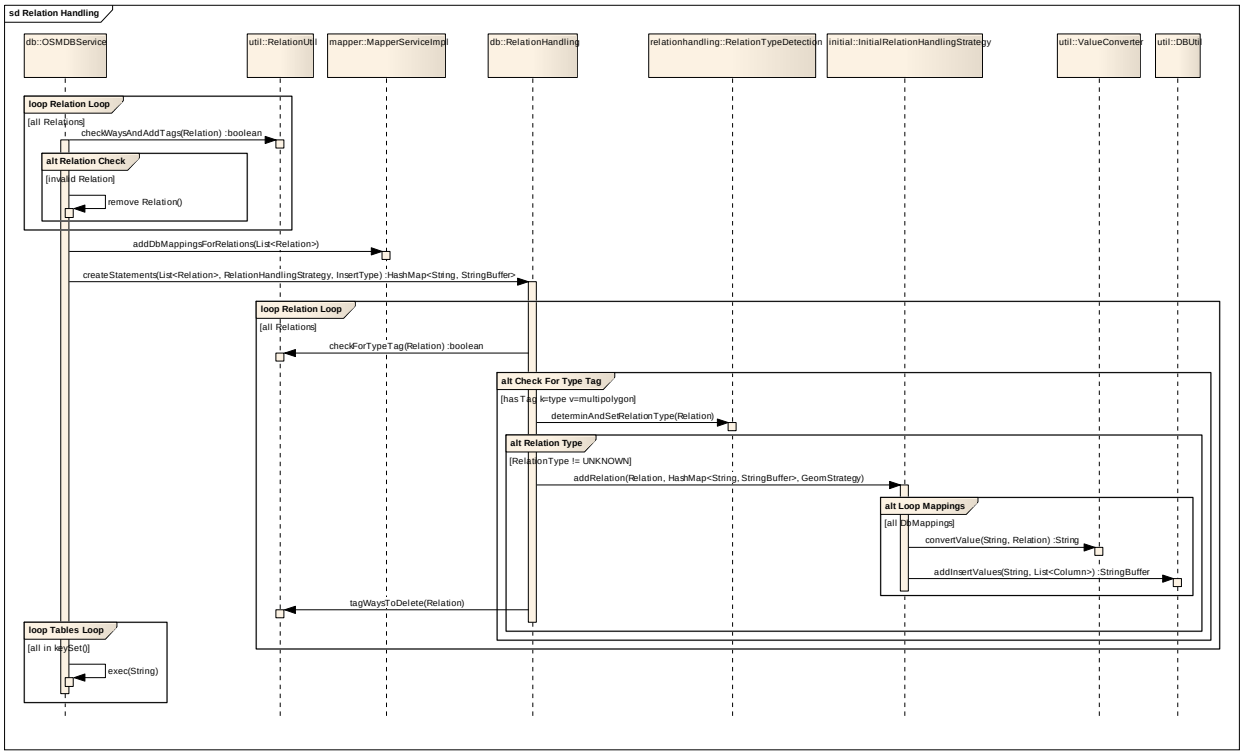


Figure 26: SD Initial Relation Handling

Explanation

7) *.importer calls on OSMDBService the method insertRelations(List<Relation>)
   a) First, each Relation needs to be checked for integrity and the Tags from its members must be added by calling the checkWaysAndAddTags method on RelationUtil.
   b) If a relation didn't pass the integrity check it's removed from the list.
   c) OSMDBService calls addDBMappingsForRelations on MapperServiceImpl which adds the mapping information from the Schema Mapping File to all relations in the list. See also the according chapter.
   d) Afterwards the method createStatements on RelationHandling is called. As HandlingStrategy we pass the InitialRelationHandlingStategy, because an initial import has to be handled. InsertTyp is: INSERT
8) RelationHandling
   a) The RelationHandling class loops over all relations in the list and removes any relation which doesn't have a Tag type=multipolygon in its tag-list.
   b) It then calls the determinAndSetRelationType method on the RelationTypeDetectionClass to find out which GeomStrategy class is needed for adding the sql.
   c) Now the addRelation-method of the InitialRelationHandlingStrategy class is called.
9) InitialRelationHandlingStrategy
   a) Initializes the SQL statement (if it is not already done by a previous call)
   b) Loops over all DbMappings found on this Relation (A Relation can be inserted in multiple tables if it matches their criteria).
   c) Loops over all Columns defined in the specific DbMapping.
   d) Converts the value from the DbMapping into the "real" value (see MapperService Chapter) by calling the convertValue-method for each Column.
   e) Adds the converted value to the sql String by passing the Column which contains the table name and all values to the DBUtil class.

10) RelationHandling
    a) Before it passes back the HashMap with all tables sql-INSERT queries, all referenced ways by this relation are tagged in the polygon_temp table as deletable.
    b) After adding all relations to their according StringBuffers in the HashMap, a semicolon will be added at the end of every statement.
11) OSMDBService now loops over the values of the resulting HashMap and executes every SQL Statement.

3.2.4.3 AREA HANDLING

3.2.4.3.1 FUNCTIONALITY

Scroll over whole polygon_temp table and search for closed ways (first and last referenced nodes are equal). Create Multipolygon-Relations out of them and insert them into the database.

3.2.4.3.2 INVOLVED PACKAGES

Packages	Explanation
ch.hsr.osminabox.db	Entry point with method handleRestlyAreas()
ch.hsr.osminabox.db.sql	Creation of area values
ch.hsr.osminabox.db.util	Util package, used for HashMap initialization, AreaHandlingUtil
*****	All RelationHandling specific Packages

3.2.4.3.3 IMPLEMENTATION

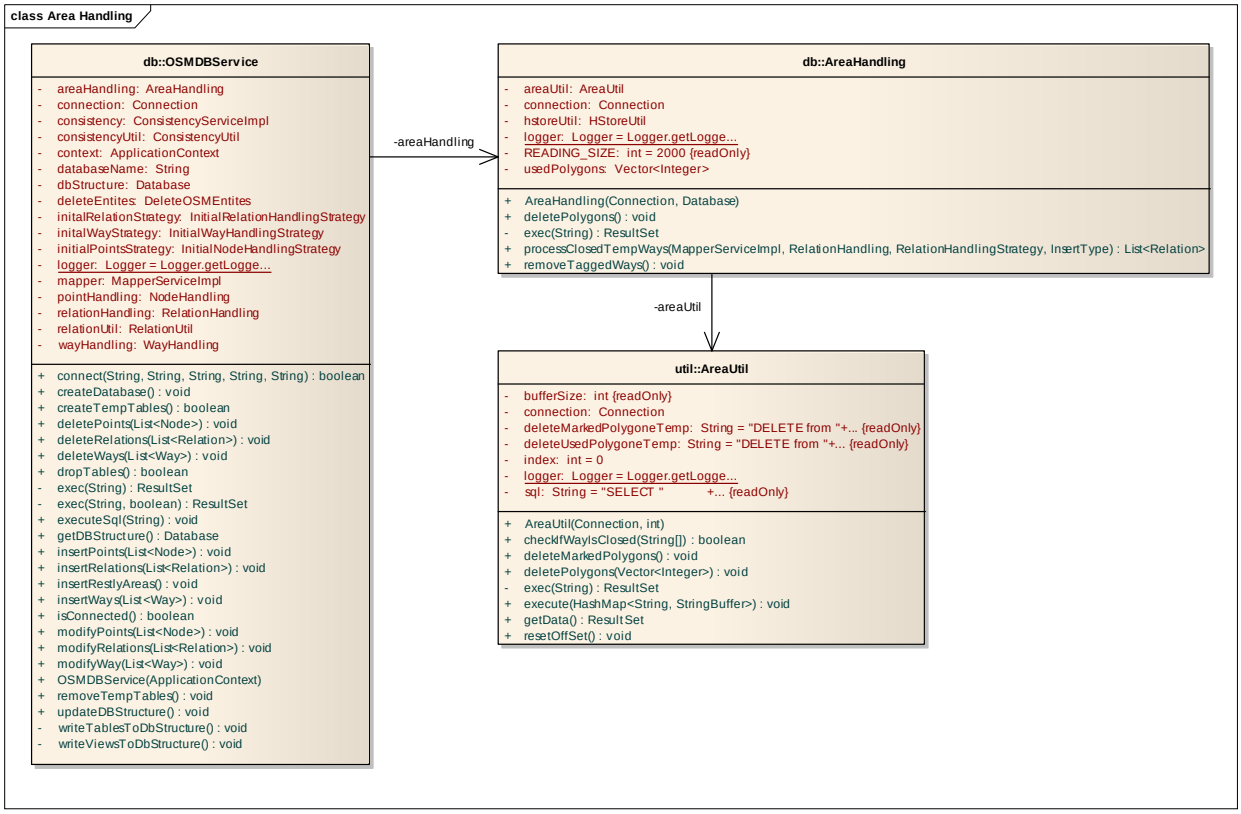


Figure 27: Area Handling

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

3.2.4.3.4 SEQUENCE DIAGRAM

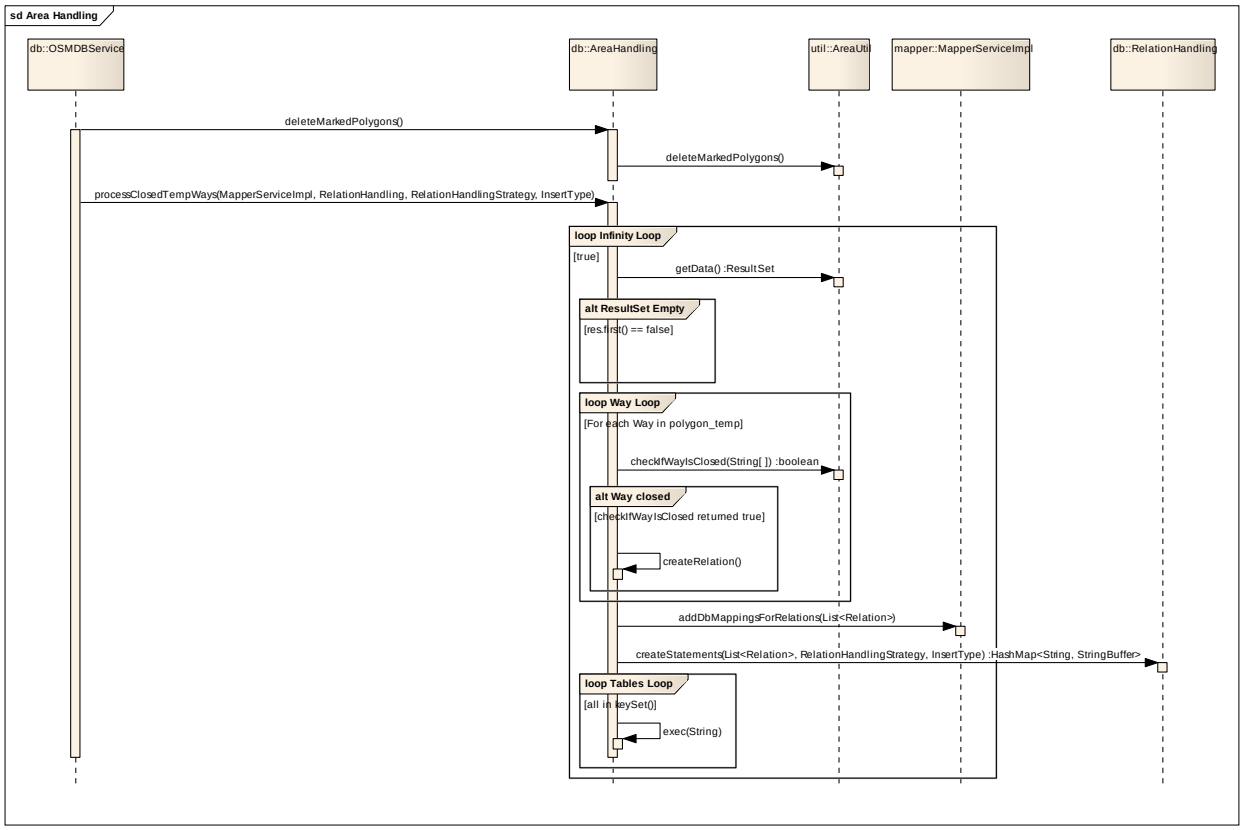


Figure 28: SD Initial Area Handling

Explanations

12) *.importer calls on OSMDBService the method insertRestlyAreas()
    a) First, all ways in the polygon_temp table which have been referenced by any inserted Relations are deleted by calling the deleteMarkedPolygon method on the AreaHandling class.
    b) Then a special method, the processClosedTempWays is called on the AreaHandling class. Important parameters are an instance of the MapperServiceImpl, the RealtionHandling and the RelationHandlingStrategy.
13) AreaHandling
    a) Within an infinity loop, the AreaHandling class asks for a certain amount of entries from the polygon_temp table by calling the getData function from AreaUtil.
    b) The infinity loop is ended if there are no more entries in the polygon_temp table.
    c) Else, each received Way is processed and checked, if it's closed (first and last nodes are the same). If not, it's deleted from the polygon_temp table.
    d) A closed Way can be treated like a Relation with one outer referenced member.
    e) A Relation Object is created with the data from the Way and the way entry deleted in the polygon_temp table.
    f) Now the same process for adding Mappings and creating sql Insert statements starts like in the insertRelations method from the OSMDBService.

3.2.4.4 CONSISTENCY SERVICE

3.2.4.4.1 FUNCTIONALITY

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

3.2.4.4.2 INVOLVED PACKAGES

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

3.2.5 IMPLEMENTATION

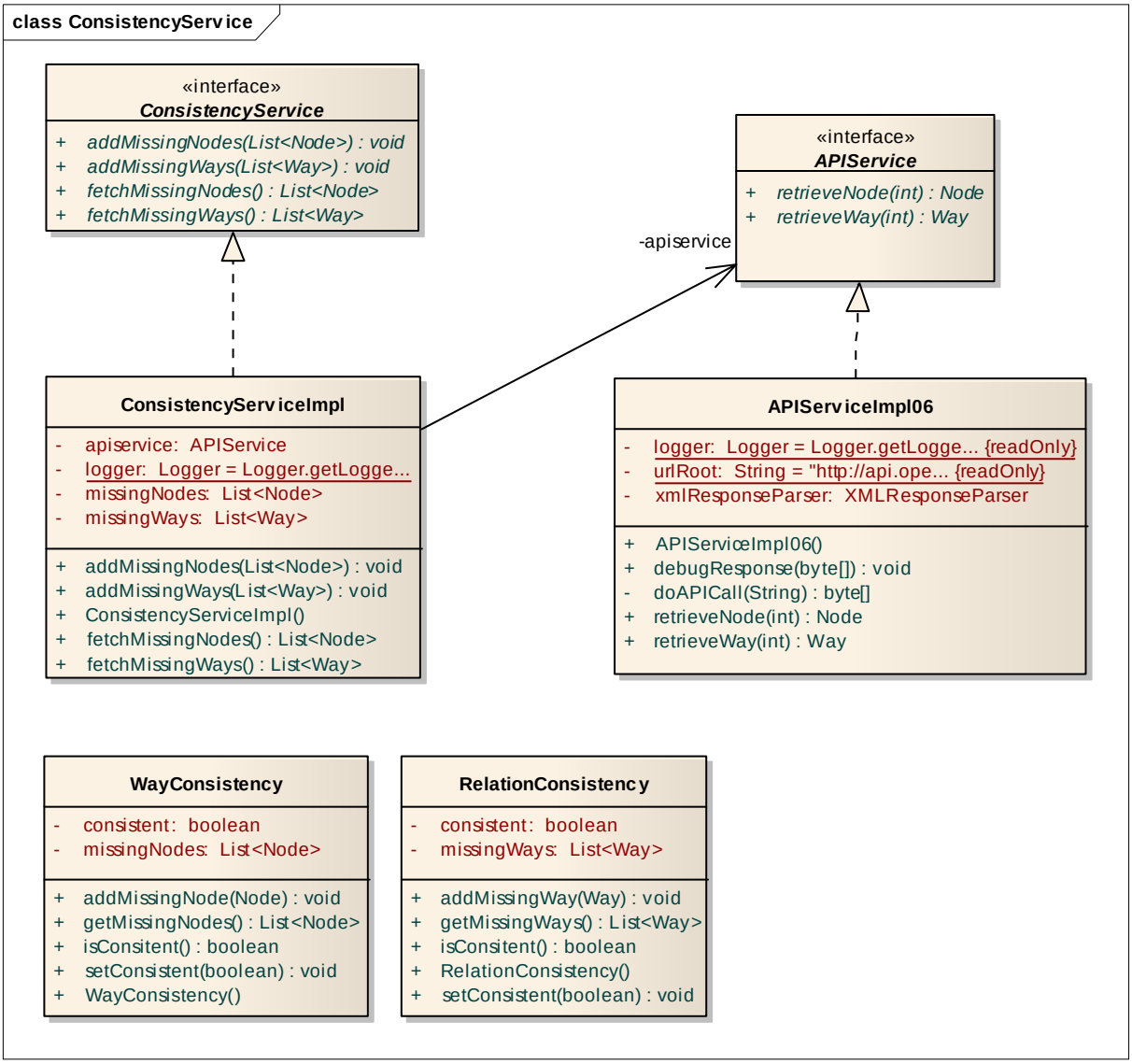


Figure 29: Class diagramm consistency service

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

3.2.5.1 DB-SCHEMA AND MAPPING HANDLING

Figure Figure 30: Overview about the Schema Mapping configuration shows the physical architecture of the way to configure OpenStreetMap-in-a-Box.

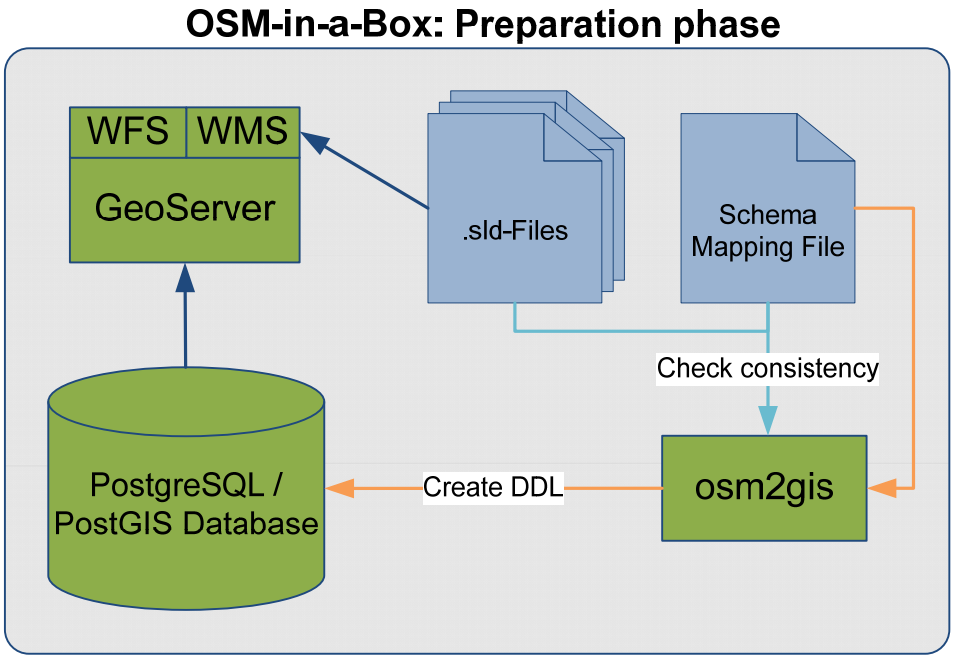


Figure 30: Overview about the Schema Mapping configuration

3.2.5.1.1 FUNCTIONALITY

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

It provides a service to generate a SQL from the mapping configuration Xml-file. This functionality is located in the package xml2ddl.

The xml2dbconsistency can check if the database in use has the same schema as this in the mapping configuration. This check is done in every initial-import or consisntency check.

The check of GeoServer SLD consistency is the third functionality.

3.2.5.1.2 INVOLVED PACKAGES

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

- xml2ddl
- xml2dbconsistency
- geoserversldconsistency

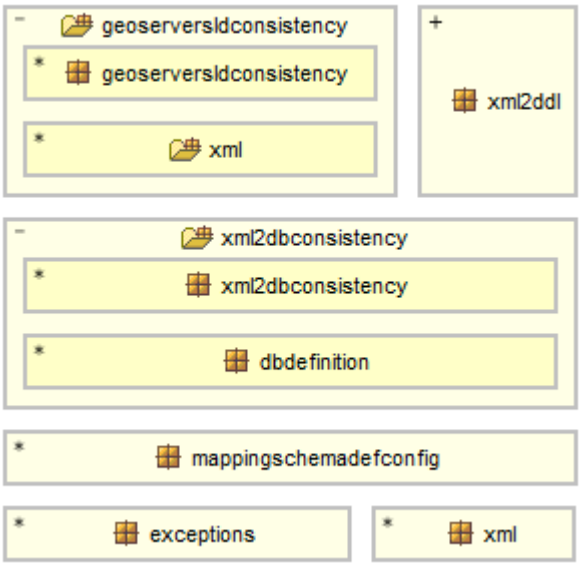


Figure 32: Package structure

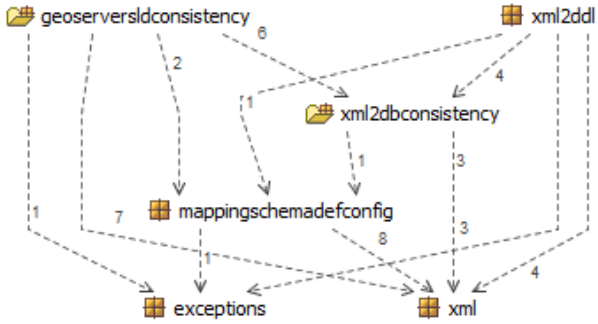


Figure 31: Dependency graph

3.2.5.1.3 INTRODUCING CONFIGSERVICE

In the package mappingschemadefconfig exist a configservice class. This class is part of the ApplicationContext and gives possibilities to read important information in the Mapping Configuration File.

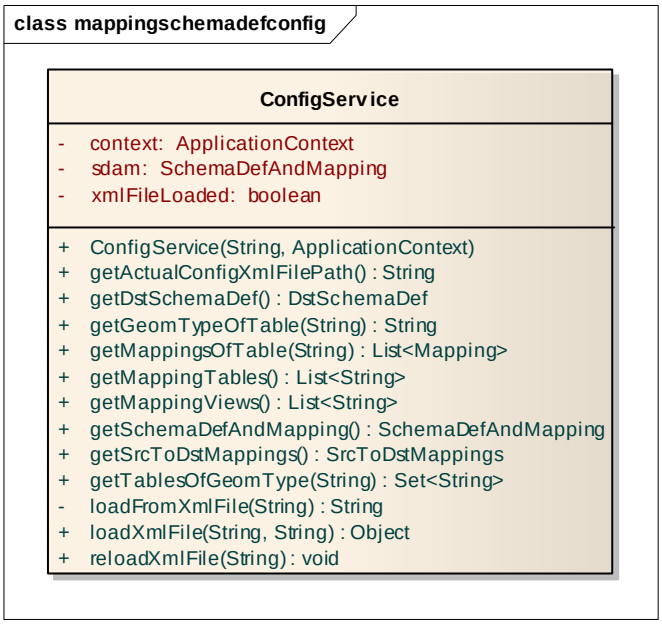


Figure 33: ConfigService

3.2.5.1.4 CONSISTENCY CHECKS

There are three consistency checks:

1. Mapping Schema File to DB
    a. This check checks if every table with his columns exists in the db in use.
2. GeoServer to DB
    a. This includes the feature types and the SLD files from the GeoServer data folder.
3. Mapping Schema File to GeoServer
    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 3 Consistency check with osm2gis –consistency.

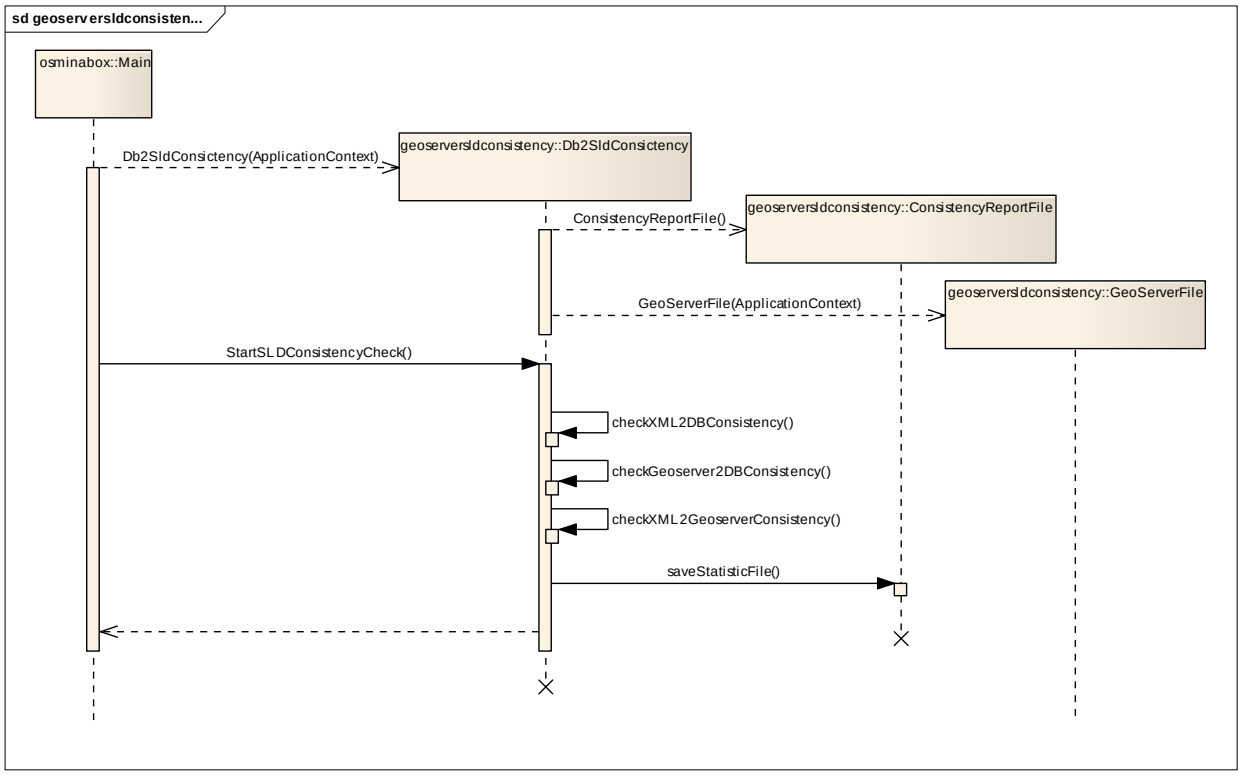


Figure 34: Sequence of consistency check

3.2.5.1.4.1 MAPPING SCHEMA FILE TO DB

The following flowchart illustrates the logic of the mapping schem file to DB consistency check:

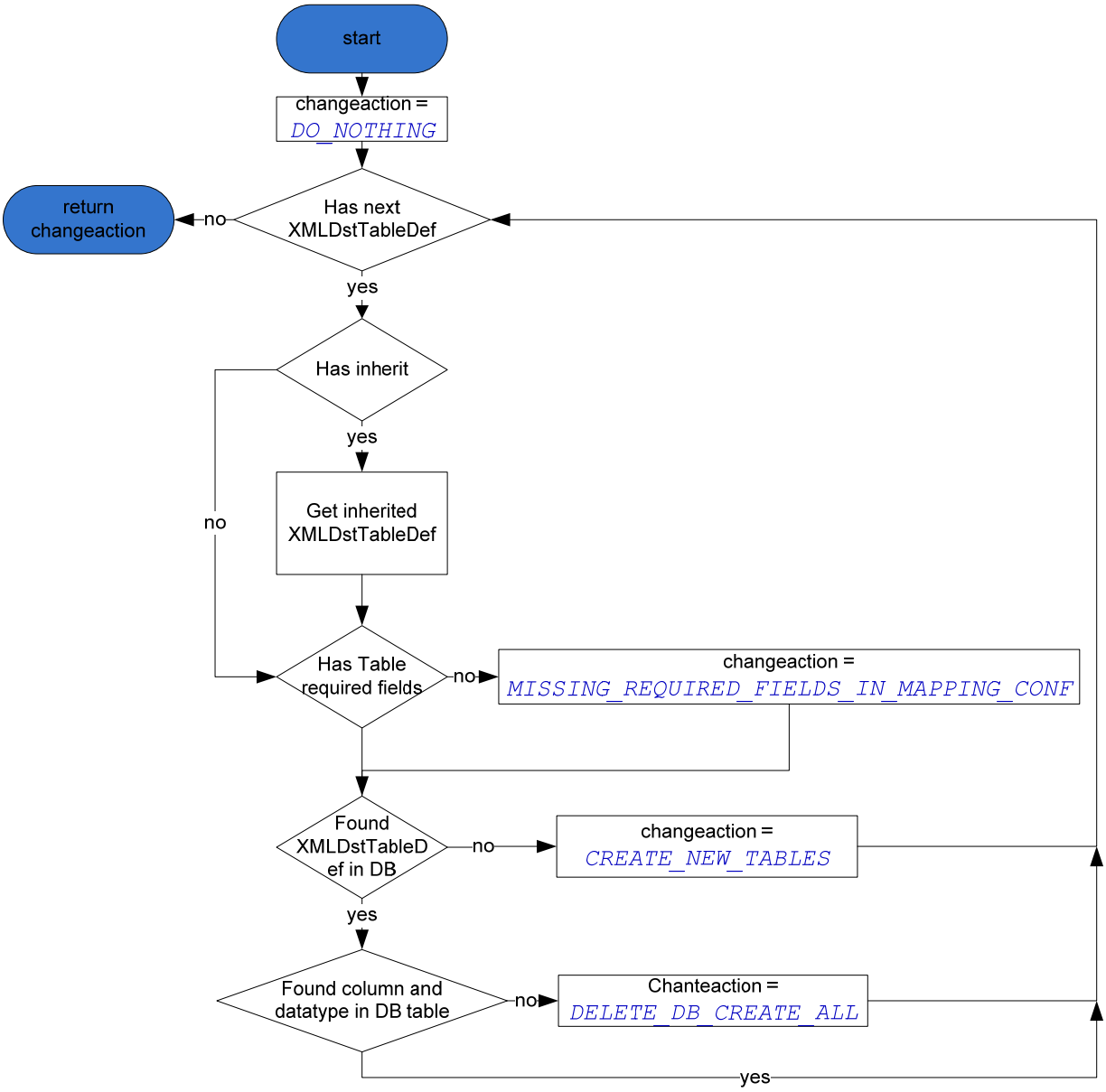


Figure 35: Flowchart mapping schema file to db

3.2.5.1.4.2 GEOSERVER TO DB

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

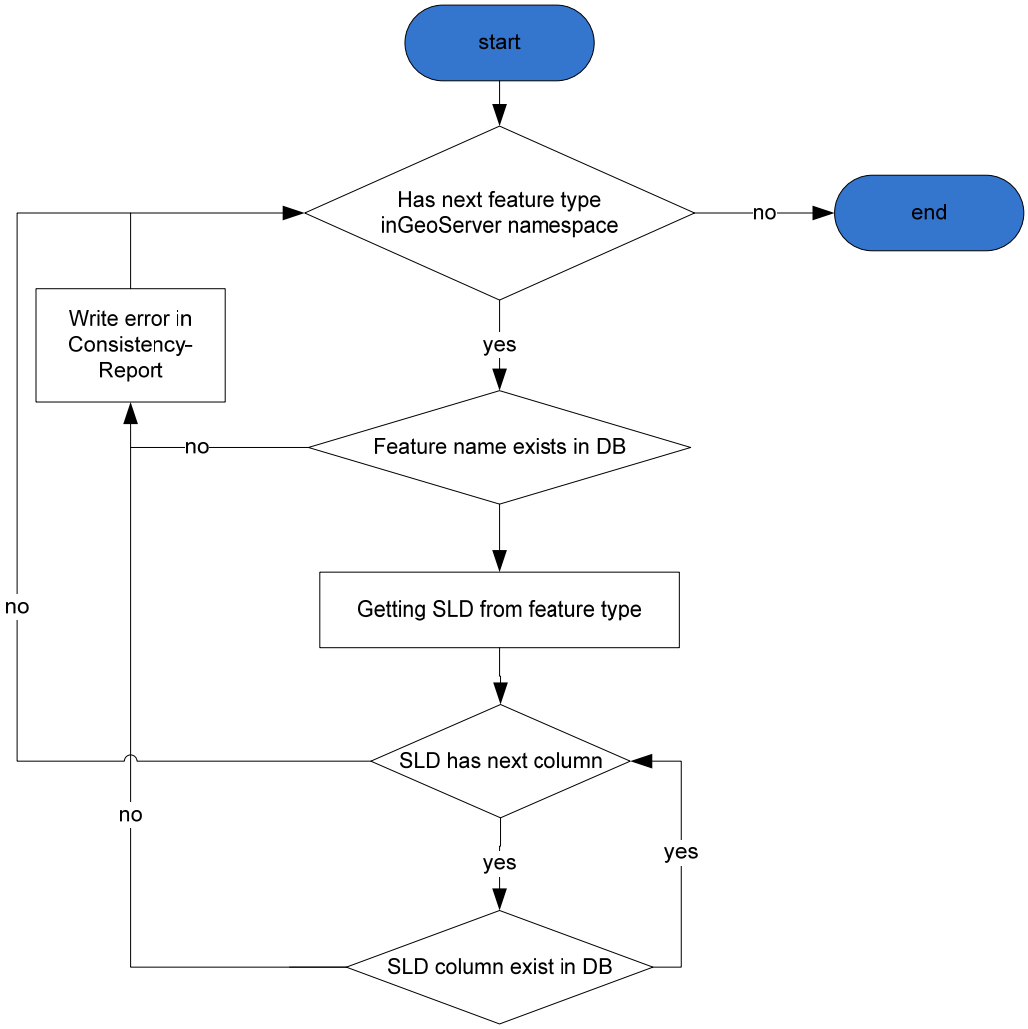


Figure 36: Flowchart GeoServer to DB

3.2.5.1.4.3 MAPPING SCHEMA FILE TO GEOSERVER

The following flowchart illustrates the logic of the mapping schem file to GeoServer consistency check:

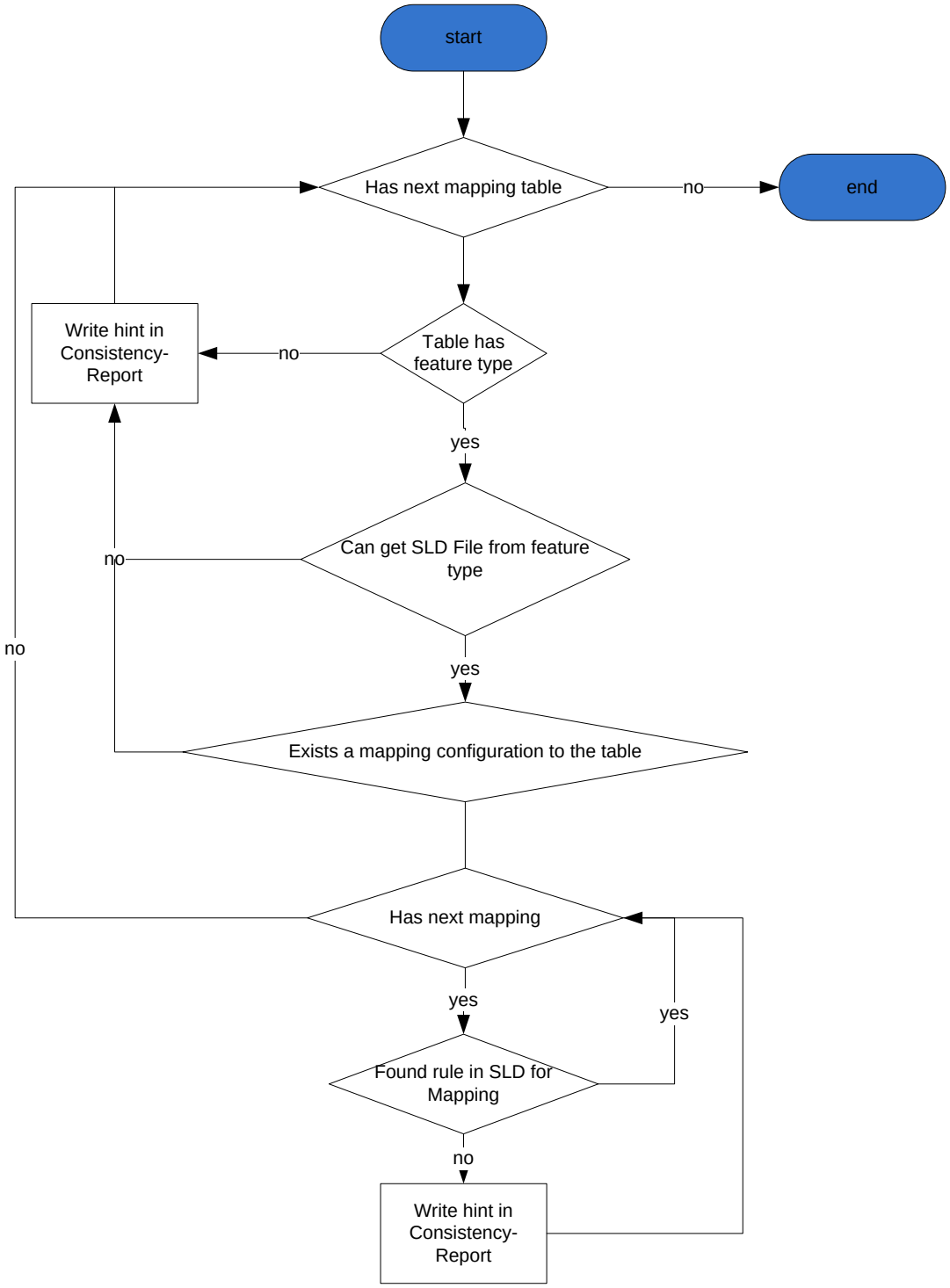


Figure 37: Flowchart mapping schema file to GeoServer

3.2.5.1.5 MAPPING SCHEMA FILE TO DDL

In every initial-import the xmldbconsistency checks if there are differences in the Schema Mapping File to the DB and if there exists a difference between this two instances it decide what should happened to solve this inconsistency.

How the check of the Schema Mapping File to the DB works is described in the chapter 3.2.5.1.4.1.

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!

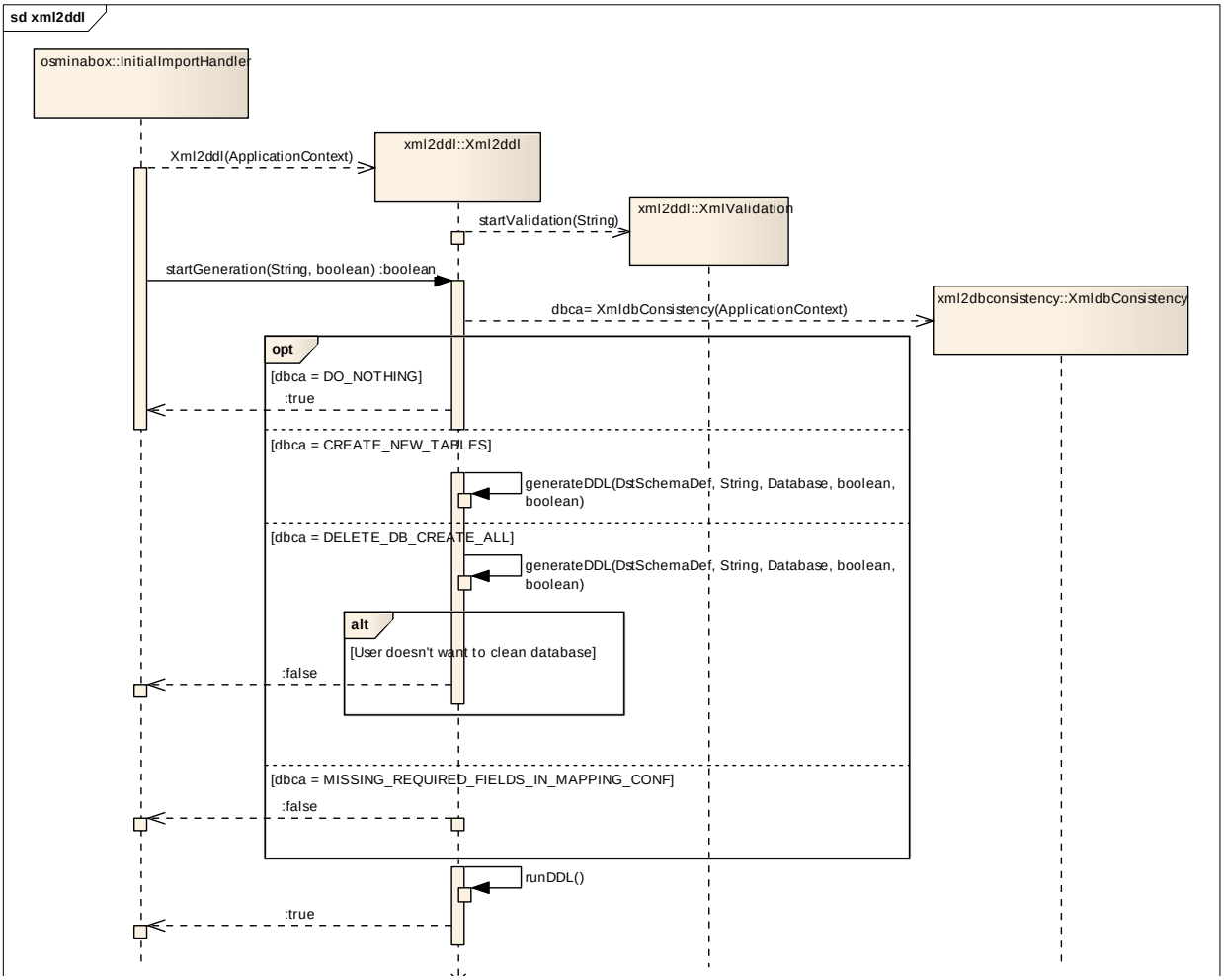


Figure 38: Sequence of start of the initial import consistency check

3.2.6 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".

3.2.6.1 MAPPING SERVICE

3.2.6.1.1 FUNCTIONALITY

The Mapping Service provides the functionality to map an OSM-Entity to a 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.

3.2.6.1.2 INVOLVED PACKAGES

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

3.2.7 IMPLEMENTATION

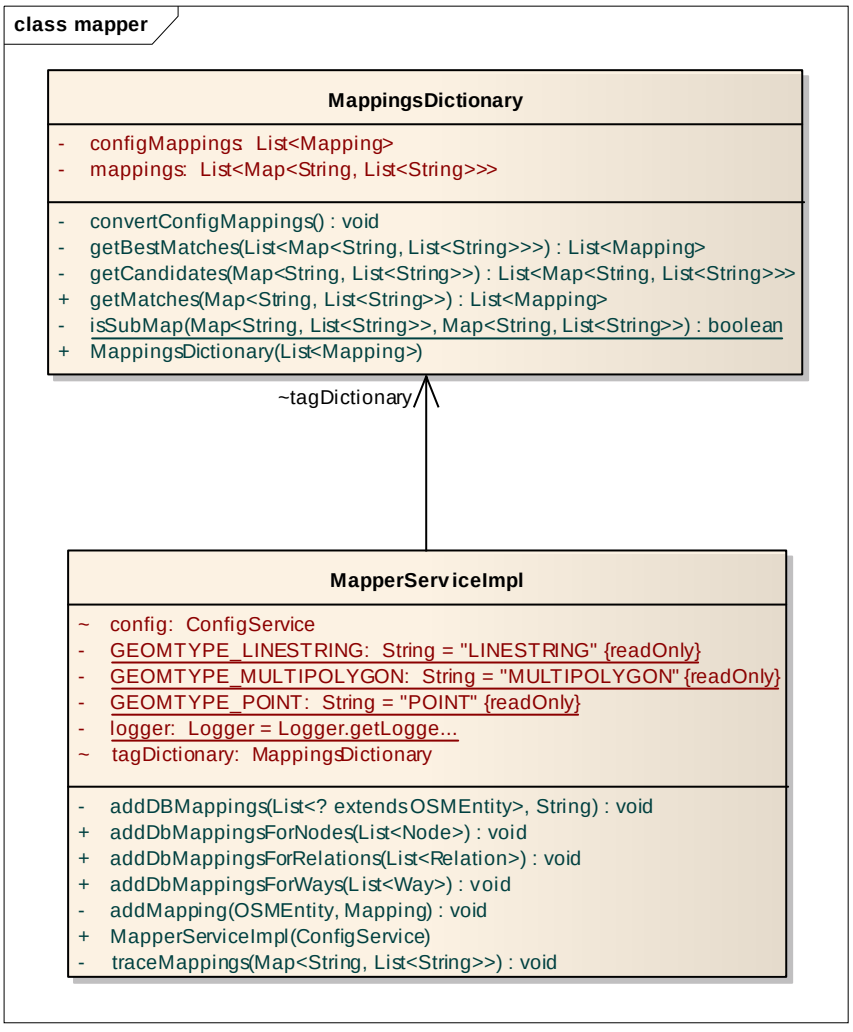


Figure 39: Mapping Service

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 the in the OSMEntity class.

3.2.8 MAPPING CONFIGURATION

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

For information about handling this file read chapter 3.2.5.1.

3.2.8.1 XML-SCHEMA

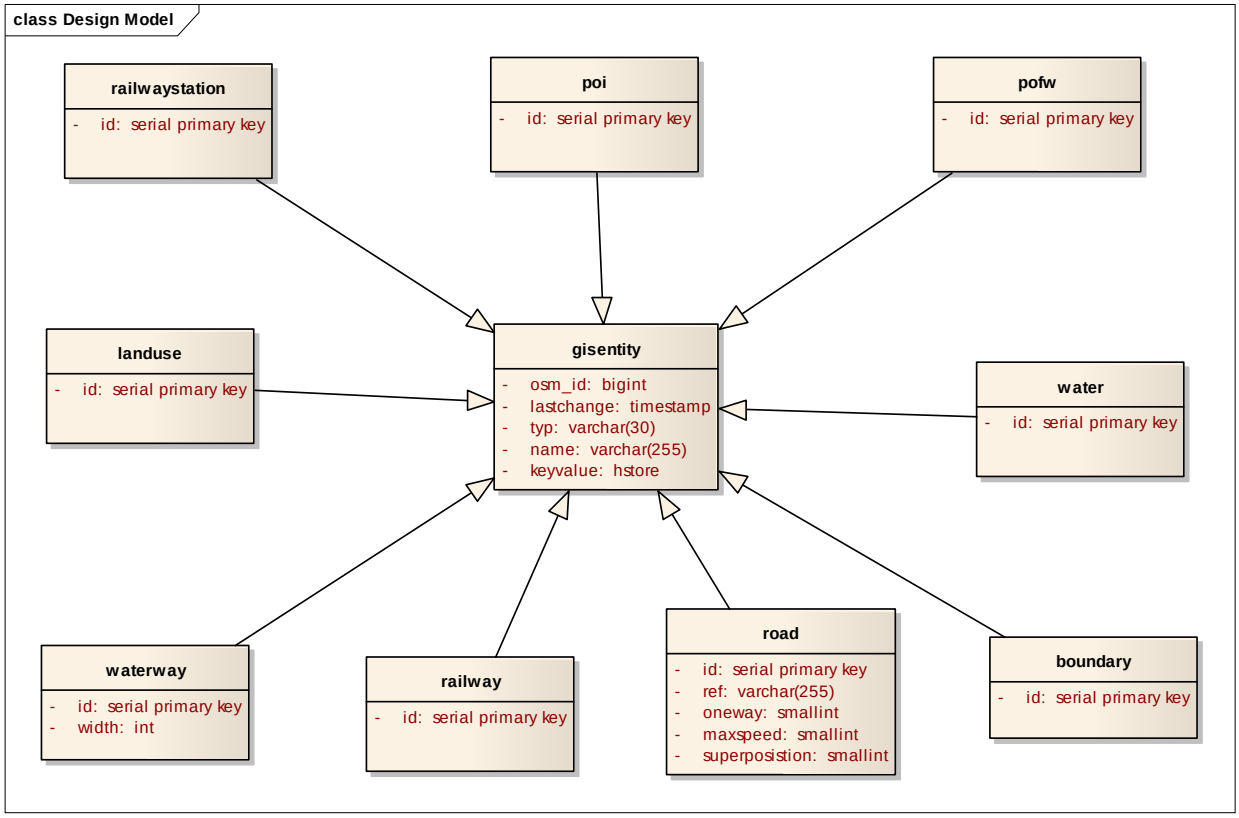
The XML-Schema is defined by the mappingconfig.xsd file that is in the config folder. Osm2gis validate this mapping configuration file on every reading.

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

3.2.9 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.

3.2.9.1 DATABASE SCHEMA



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).

3.2.9.2 DATABASE SOFTWARE

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

Software	Version	Link
PostgreSQL	8.3	http://www.postgresql.org/
PostGis	1.3.5	http://postgis.refractions.net/

PROJEKTMANAGEMENT

3.2.10 PROJEKTORGANISATION

Das Projektteam für die Semesterarbeit 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.

3.2.10.1 ORGANISATIONSSTRUKTUR

Name	**Email**	**Verantwortlichkeit**
Meier Andreas	ameier@hsr.ch	Evaluation Schema-Mapping Implementation Konsistenzprüfung
Zimmermann Joram	jzimmerm@hsr.ch	Evaluation Konsistenzprüfung Implementation Rewrite Import Converter

3.2.10.2 BETREUUNG

Name	**Email**	**Funktion**
Prof. Stefan Keller	sfkeller@hsr.ch	Betreuung

3.2.11 MANAGEMENT ABLÄUFE

3.2.11.1 ZEITMANAGEMENT

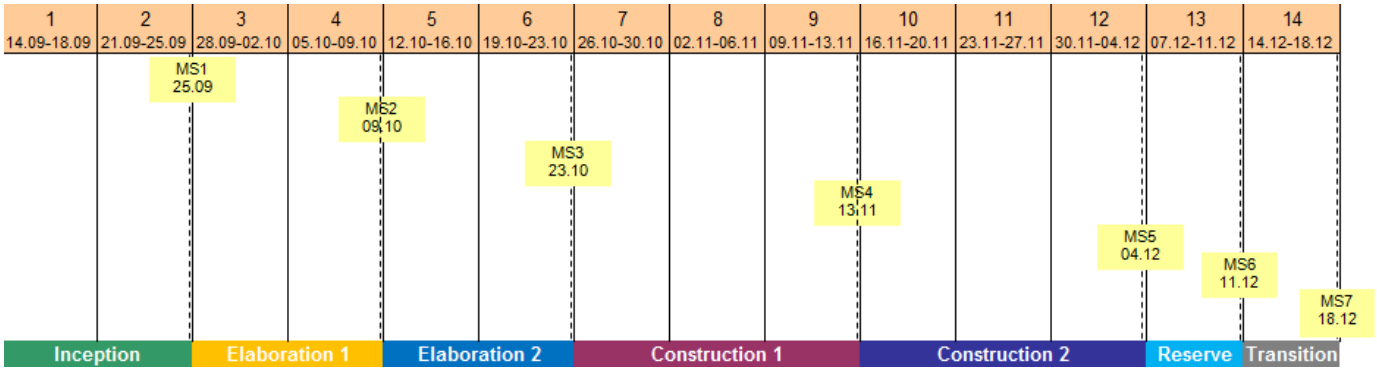
Für die gesamte Semesterarbeit stehen 14 Wochen zu Verfügung. Der Projektplan wurde so ausgelegt, dass die 14 Wochen vollständig ausgenutzt werden. Die als Reserve eingeplante Zeit in Woche 13 des Projektes wird entweder als Puffer für allfällige Verzögerungen, oder für die Implementation weiterer Features genutzt.

3.2.11.2 PROJEKTPLAN

3.2.11.2.1 ZEITPLAN

Siehe Projektplan.xlsx für mehr Details.

3.2.11.2.2 ITERATIONSPLANUNG



3.2.11.3 MEILENSTEINE

Meilenstein 1: Einarbeitung, Technikstudium	25.09.2009
• Projektplan in einer ersten Version	
Artefakte: - Projektplan - Zeitplan	

Meilenstein 2: Endgültige Aufgabenstellung, Projektplan, Requirements	09.10.2009
• Projektplan ist erstellt und für nächste Iteration vorbereitet, weitergeführt. • Anforderungen in Use Cases sind fertiggestellt. • Technikstudium ist zum grössten Teil abgeschlossen. • Lauffähige Vorgängerarbeit OpenStreetMap-in-a-Box	
Artefakte: - Projektplan - Zeitplan - Anforderungsspezifikationen - Domainanalyse	

Meilenstein 3: Prototyp 1 für DDL Generierung, Definition der Implementierungsarbeiten	23.10.2009
• Prototyp für die DDL Generierung aus dem Config-XML File ist vorhanden. • Die Arbeitspakete für die Implementation wurden erstellt.	
Artefakte: - Domainanalyse - SSD + Contracts - Software Architecture Document - Projektplan	

Meilenstein 4: Configsupport implementiert	13.11.2009
• xml2ddl Implementation beendet und integriert. • Importconfig ist lauffähig	
Artefakte: - Software Architecture Document	

Meilenstein 5: Implementation der Hauptaufgabenstellung sind beendet	04.12.2009
• Vollständiger Configsupport. • Konsistenzprüfung DB zu Geoserver.	
Artefakte: - Software Architecture Document	

Meilenstein 6: Reserve	11.12.2009
• Reserve	
Artefakte:	

Meilenstein 7: Integration des Gesamtsystems	18.12.2009
• Es steht ein Showcaseserver im Netz zur Verfügung. • Abgabe	
Artefakte: - Installationsanleitung	

3.2.12 ARBEITSPAKETE

3.2.12.1 PROJEKTMANAGEMENT

5.1.1 Projektplan Excel		18 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		22 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		7 Std
Beschreibung	Die persönliche Zeiterfassung für das Projekt.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.1.4 Review Projektplan		5.4 Std
Beschreibung	Review des Projektplans Excel und Word.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.1.1 Projektplan Excel 5.1.2 Projektplan Dokument	

3.2.12.2 REQUIREMENTS

5.2.1 Anforderungsspezifikation		4 Std
Beschreibung	Erstellen der 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		16 Std
Beschreibung	Erarbeiten der Use Cases die für die Anforderungsspezifikation von belangen sind.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.2.3 Review Anforderungsspezifikation		0.5 Std
Beschreibung	Review der Anforderungsspezifikation	
Verantwortlich	ameier	
Abhängigkeiten	5.2.1 Use Cases	

3.2.12.3 ANALYSE

5.3.1 Analyse Vorgängerprojekt		60 Std
Beschreibung	Erarbeitung der Domainanalyse bezüglich des Full-Config Supports.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.3.1 Domainanalyse		6 Std
Beschreibung	Erarbeitung der Domainanalyse bezüglich des Full-Config Supports.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.3.2 SSD + Contracts		3 Std
Beschreibung	Beschreibung gewisser SSD anhand der Anforderungsspezifikation. Beschreibung von Contracts bezüglich der Domainanalyse.	
Verantwortlich	jzimmerm	
Abhängigkeiten	5.2.1 Anforderungsspezifikation 5.3.1 Domainanalyse	

5.3.3 Review Domainanalyse		0.6 Std
Beschreibung	Review der Domainanalyse	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.3.1 Domainanalyse	

5.3.4 Evaluation Config-Strategie		8 Std
Beschreibung	Evaluierung wie die Config-Strategie umgesetzt werden soll.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.3.1 Domainanalyse	

5.3.5 Evaluation Konsistenzprüfung Grafikkonfiguration Geoserver		6 Std
Beschreibung	Evaluierung wie die Konsistenzprüfung der Grafikkonfiguration des Geoservers umgesetzt werden soll.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.3.1 Domainanalyse 5.3.4 Evaluation Config-Strategie	

3.2.12.4 DESIGN ANALYSE

5.4.1 Softwarearchitektur Dokument		31 Std
Beschreibung	Die Softwarearchitektur wird in einem Dokument erfasst. Dazu gehören unter anderem die Erklärung der Packetstruktur, der Klassen und aussagekräftige Sequenzdiagramme.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.4.2 Softwarearchitektur Prototyp für DDL		18 Std
Beschreibung		
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.3.4 Evaluation Config-Strategie 5.3.5 Evaluation Datenbankstruktur	

5.4.3 Softwarearchitektur Review		2 Std
Beschreibung	Review der Softwarearchitektur	

Verantwortlich	ameier / jzimmerm
Abhängigkeiten	5.4.1 Softwarearchitektur Dokument

3.2.12.5 IMPLEMENTATION

5.5.1 xml2ddl Integration	10 Std
Beschreibung	xml2ddl Integration
Verantwortlich	ameier
Abhängigkeiten	-

5.5.2 xml2ddl Erweiterungen	13 Std
Beschreibung	xml2ddl Erweiterungen
Verantwortlich	ameier
Abhängigkeiten	-

5.5.3 Parser Refactoring	23 Std
Beschreibung	Parser Refactoring
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.5.3 Rewrite Import Konverter	44 Std
Beschreibung	Rewrite Import Konverter
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

5.5.4 Konsistenzprüfung Grafikkonfiguration Geoserver	20 Std
Beschreibung	Konsistenzprüfung Grafikkonfiguration Geoserver
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

5.5.5 Integration der Konfigurationen von Teil B	10 Std
Beschreibung	Integration der Konfigurationen von Teil B
Verantwortlich	ameier
Abhängigkeiten	-

5.5.6 Integration der Website (- Erweiterungen) von Teil B	10 Std
Beschreibung	Integration der Website (- Erweiterungen) von Teil B
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.5.6 Installer	20 Std
Beschreibung	Installer
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

5.5.7 MapCompare	5 Std
Beschreibung	MapCompare
Verantwortlich	jzimmerm
Abhängigkeiten	-

3.2.12.6 QUALITÄTSMASSNAHMEN

5.6.1 Technikstudium	8 Std
Beschreibung	Einlesen in die verschiedenen zu verwendenden Technologien.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

5.6.2 Risikomanagementdokument	4 Std
Beschreibung	Erstellen des Risikomanagement.
Verantwortlich	ameier / jzimmerm
Abhängigkeiten	-

3.2.12.7 DOKUMENTATION

5.7.1 Abstract	2 Std
Beschreibung	Abstract
Verantwortlich	jzimmerm
Abhängigkeiten	-

5.7.2	1 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	6 Std
Beschreibung	Benutzer- und Installationshanbuch
Verantwortlich	ameier
Abhängigkeiten	-

5.7.8 Plakat	5 Std
Beschreibung	Plakat
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

5.7.9 Gesamtdokumentation	28 Std
Beschreibung	Gesamtdokumentation
Verantwortlich	ameier/jzimmerm
Abhängigkeiten	-

3.2.12.8 SITZUNGEN

5.8.1 Wöchentliche Sitzungen		42 Std
Beschreibung	1.5 h pro Woche	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	-	

5.8.2 Sitzungsvorbereitungsprotokoll		6.5 Std
Beschreibung	Sitzungsvorbereitungsprotokoll erstellen. Das jeden Mittwochabend Herr Keller gesendet wird.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.8.1 Sitzungen	

5.8.3 Sitzungsprotokoll		7 Std
Beschreibung	Sitzungsprotokoll erstellen.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.8.1 Sitzungen	

5.8.4 Sitzungsprotokoll Review		7 Std
Beschreibung	Review des Protokolls.	
Verantwortlich	ameier / jzimmerm	
Abhängigkeiten	5.8.3 Sitzungsprotokoll	

3.2.13 RISIKO MANAGEMENT

Risiko Kosten OpenStreetMap-in-a-Box A

Risiko Analyse
Projektname: OpenStreetMap-in-a-Box A **Projektmanager: Meier Andreas, Zimmermann Joram** **Datum Kalkulation: 01.10.2009**

				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	**Einarbeitungszeit 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	**Teilarbeit B wird nicht rechtzeitig fertiggestellt**	**Keine Integration möglich**	**Eigenständiger Showcase**	**20**	50	20%	**10**	**Mittel**
R06								
	Total Kosten in Arbeitspaketen enthalten			**200**				
	Total Rückstellungen						**35**	

3.2.14 INFRASTRUKTUR

Als Infrastruktur dienen uns folgende Komponenten:

- Linux Server:
  - http://sinv-56018.edu.hsr.ch
    - Tomcat 5.5, Port 8080
    - PostGreSQL, Port 5432
- 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/

3.2.14.1 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.

3.2.14.2 ENTWICKLUNGSSOFTWARE VERSION

Die Entwicklung wird mit folgender Software durchgeführt:

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

3.2.15 QUALITÄTSMASSNAHMEN

3.2.15.1 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)

3.2.15.2 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.

3.2.15.3 TESTPLANUNG

3.2.15.3.1 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.

3.3 PROJEKTMONITORING

3.3.1 SOLL-IST-ZEIT-VERGLEICH

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

Überpakete	Soll	Ist	Differenz
Projekt Management	52.4	20.9	31.5
Requirements	20.5	8.0	12.5
Analyse	83.6	124.0	-40.4
Design Analyse	51.0	39.0	12
Implementation	155.0	199.0	-44.0
Qualitätsmassnahmen	12.0	9.0	3.0
Dokumentation	46.0	31.0	15.0
Sitzungen	62.5	72.5	-10

3.3.2 CODESTATISTIK

Die folgenden Codestatistiken wurden mit Structure101 generiert:

Size	
Jars (and / or classpath directories)	1
Packagages (that contain classes)	41
Classes	299
NI(Number of bytecode Instructions)	26’000
LOC(Non Comment Non Blank Lines of Code)	11’000

Term	Threshold	#Offenders	Offenses (%)	XS contribution
Tangled (design)	0	2 of 11	18%	39%
Fat (design)	120	0 of 11	0%	0%
Fat (leaf package)	120	1 of 41	2%	61%
Fat (class)	120	0 of 299	0%	0%
Fat (method)	15	0 of 693	0%	0%
Total			20%	100%

3.3.3 SITZUNGSPROTOKOLLE

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

3.4 BENUTZERDOKUMENTATION

3.4.1 INSTALLATION

3.4.1.1 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 1.4.0 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/).

3.4.1.2 STEP 1 CREATE A DATABASE

Create a database with the PostGIS template.

3.4.1.2.1 ON WINDOWS SYSTEMS

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

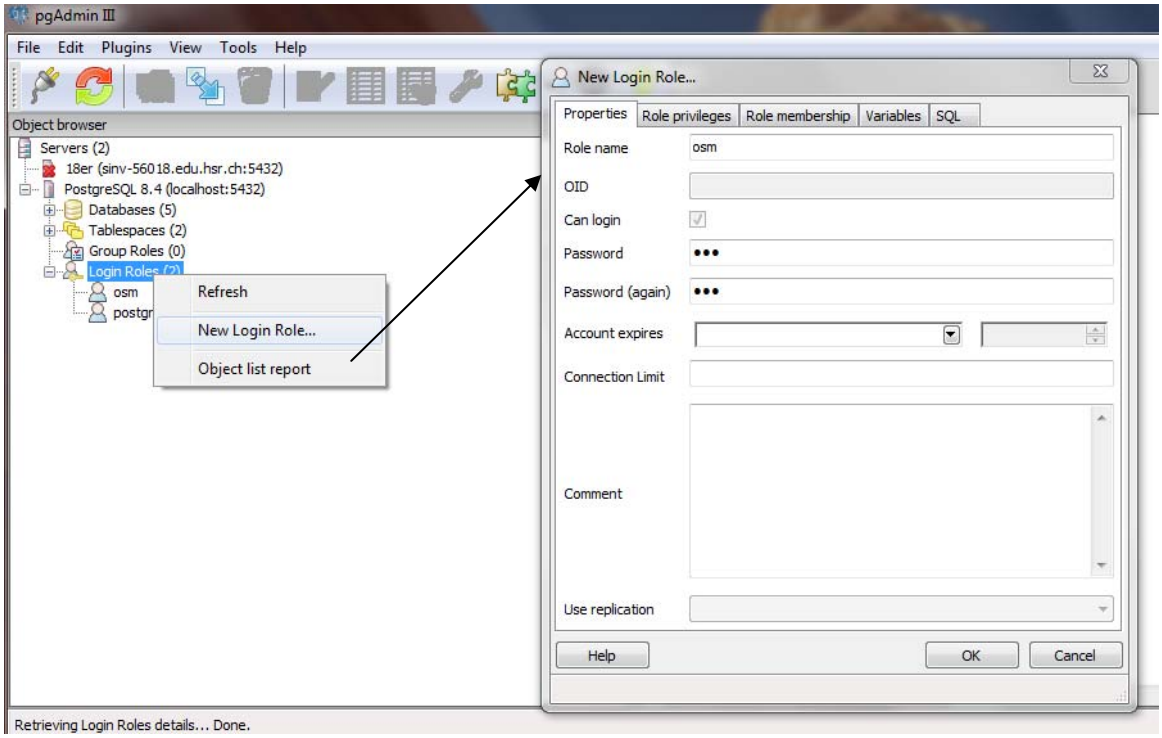
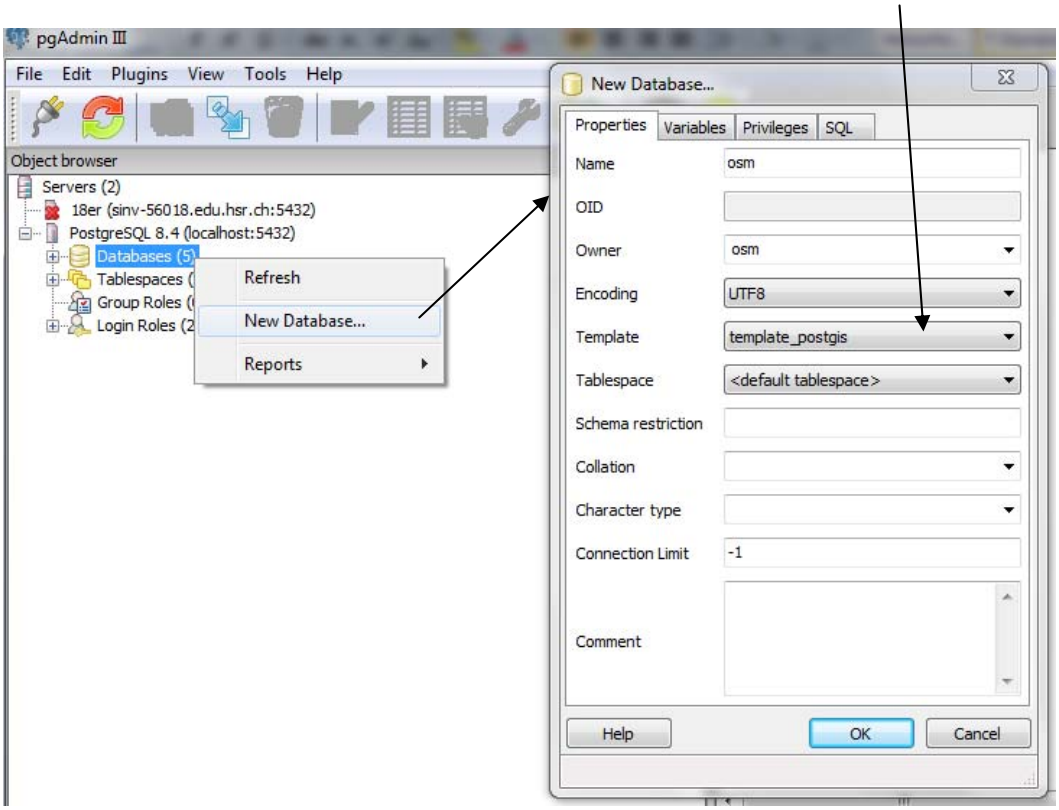
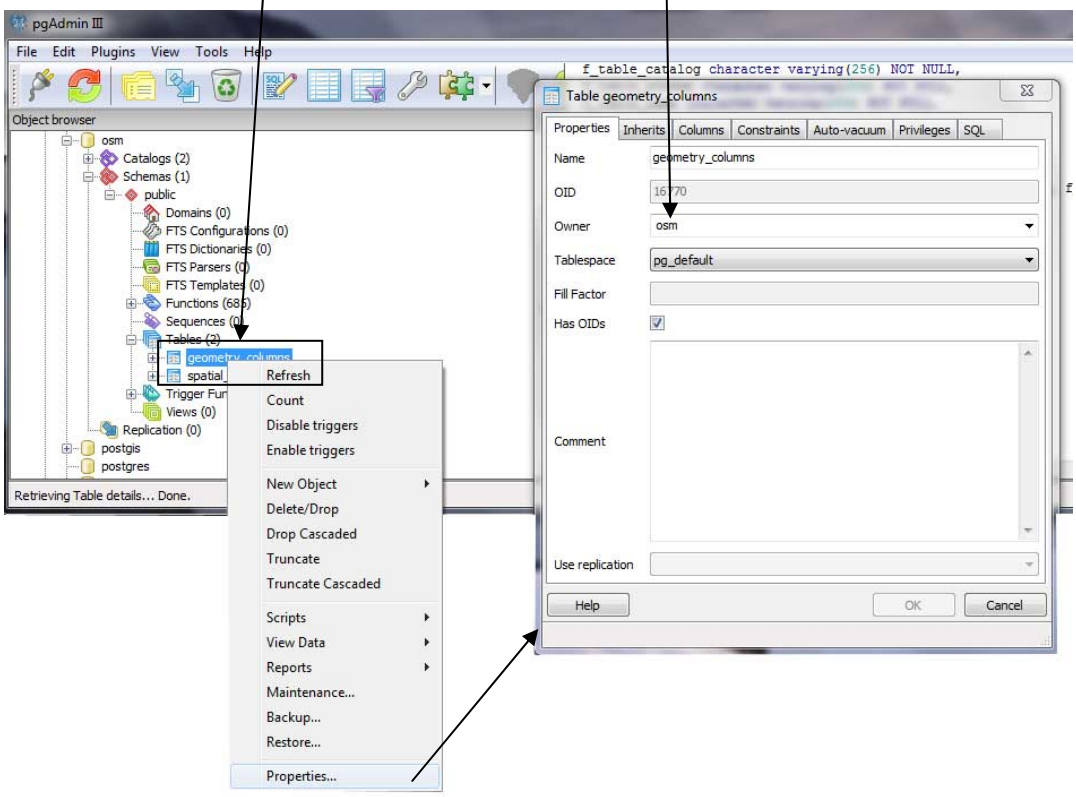


Figure 40: Creating a new role

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



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



3.4.1.2.2 ON UNIX SYSTEMS

1. Flex library must be installed.
2. Download the source of PostGIS
3. Unpack source tar xvzf postgis-**.tar.gz (** is the version of your PostGIS)
4. Go into the new folder you unpacked
5. Run *./configure*
6. Run *make*
7. Run *make install*

Now we have installed PostGIS.

Creating a Postgres db with PostGIS template

```
createdb yourtestdatabase
createlang plpgsql yourtestdatabase
psql -d yourtestdatabase -f postgis.sql
psql -d yourtestdatabase -f spatial_ref_sys.sql
```

3.4.1.3 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 3.4.4.

3.4.2 MAPPING CONFIGURATION

There exists already a mapping configuration in your installation, in config/mappingconfig.xml.

This is the standard mapping configuration of the OpenStreetMap-in-a-Box and 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 you installation, more information about this see chapter 3.4.6.

3.4.2.1 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 41: Base schema of mapping configuration

3.4.2.2 CONFIGURING DESTINATION SCHEMA

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

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

    <dst_table_def_user_defined>
        <!-- user defined SQL -->
        <![CDATA[
            CREATE SEQUENCE pois_myserial_seq;
        ]]>
    </dst_table_def_user_defined>

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

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

Figure 42: Configuring destination schema

3.4.2.2.1 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

- osm_id
- geom

```
<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 43: Defining tables

3.4.2.2.1.1 THE GEOMETRY DATATYPE

Every table must have a geometry type. This can be declared with *type="geometry(4326, 'MULTIPOLYGON', 2)"* with this information the osm2gis will generate the SQL *Select AddGeometryColumn('water', 'geom', 4326, 'MULTIPOLYGON', 2);*

3.4.2.2.2 DEFINING USER DEFINED SQL <DST_TABLE_DEF_USER_DEFINED>

If you have to use other SQL commands that are not defined you can use <dst_table_def_user_defined>.

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

Figure 44: Defining user defined SQL

3.4.2.2.3 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 45: Defining views

If you define views, you have to do an insert into geometry_columns to register your view that geoserver know this view.

3.4.2.3 CONFIGURING SOURCE SCHEMA MAPPING

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.

```
<mapping>
    <and_ed_conditions>
        <nodes_tags k="amenity" v="hotel" />
    </and_ed_conditions>
    <dst_table name="poi" />
    <dst_columns>
        <column name="osm_id" value="%attribute_id%" />
        <column name="lastchange" value="%attribute_timestamp%" />
        <column name="typ" value="hotel" />
        <column name="name" value="%tag_name%" />
        <column name="keyvalue" value="%tags_all%" />
        <column name="geom" value="%geom%" />
    </dst_columns>
</mapping>
```

Figure 46: Mapping configuration

3.4.3 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%”/>

3.4.4 INITIAL IMPORT

The import can be executed using the following options:

3.4.4.1 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.

3.4.4.2 IMPORT THE PLANET FILE FROM A LOCAL PATH

osm2gis --initial-import -l [planetfile]

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

3.4.4.3 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.

3.4.4.4 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

3.4.5 DIFFERENTIAL UPDATE

Differential update is not supported in Version 2.0.

3.4.6 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.

3.4.6.1 STRUCTURE OF CONSISTENCY REPORT

At first you should know the architecture of the configuration.

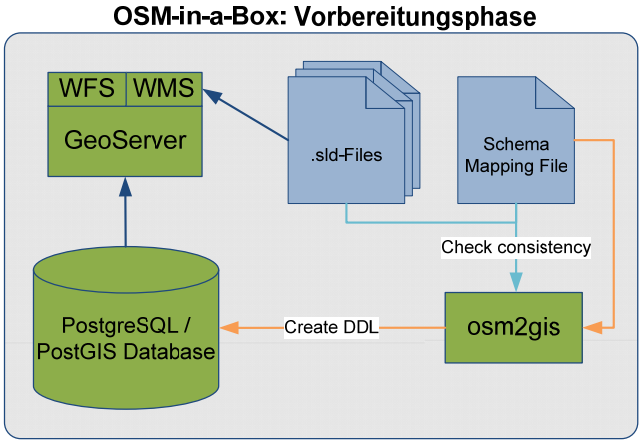


Figure 47: Configuration

3.4.6.1.1 SUMMARY

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

3.4.6.1.2 DETAILS

3.4.6.1.2.1 TABLES DEFINED IN SCHEMA MAPPING FILE TO DB IN USE

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!
```

3.4.6.1.2.2 CHECKING GEOSERVER CONFIGURATION 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!
------------------------------------------------------------------
------------------------------------------------------------------
```

3.4.6.1.2.3 CHECKING MAPPING CONFIGURATION TO SLD FILES

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!
------------------------------------------------------------------
```

3.4.7 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]

3.4.8 HELP

Help inside the command line can be found by typing:

osm2gis --help

4 ANHANG

4.1 ERFAHRUNGSBERICHTE

4.1.1 ANDREAS MEIER

4.1.1.1 TEAM

Die Zusammenarbeit mit Joram Zimmermann hat sehr gut geklappt. Da wir schon die Arbeiten in User Interfaces 1 und Software Engineering 2 zusammen bewältigt haben wissen wir wie der andere denkt und wo die Stärken und Schwächen des Gegenübers liegen. Auch die Aufgaben konnten wir, sobald klar war was wir machen mussten, gut verteilen.

Durch unsere ähnlichen Stundenpläne konnten wir die Zeiten in denen wir Zusammenarbeiten mussten gut einteilen.

Auch die Zusammenarbeit mit Herr Marco Busarello von der Diplomarbeit Teil B ist unkompliziert und professionell über die Bühne gegangen.

4.1.1.2 ZEITPLANUNG

Die Einarbeitung in das bestehende Projekt haben wir ganz klar unterschätzt. Dies nahm sehr viel Zeit in Anspruch. Als jedoch ein Überblick über das Projekt vorhanden war, kam ich mit dem Programmieren der Konsistenzprüfung und dem Präprozessor, der das DDL aus dem Schema Mapping File generiert, gut und fast ohne Zwischenfälle voran.

Gegen Ende der Arbeit wurde uns bewusst das wir für die Integration in die vorhandene Serverumgebung zu wenig Zeit haben und unsere ganze Reservezeit bereits aufgebraucht war. Zudem ist mein Wissen über das eingesetzte Server-OS zu klein um das in kurzer Zeit lösen zu können. Darum konnten wir leider auch keinen lauffähigen Server zu Verfügung stellen.

4.1.1.3 NEU ERLERNTE TECHNOLOGIEN

- JAXB
- OSM-Daten
- GeoServer

4.1.1.4 FAZIT

Meiner Meinung nach haben wir die Arbeit ganz gut bewältigt. Das Thema war spannend zu erarbeiten. Auch war es einmal interessant sich in ein bereits bestehendes Projekt einzuarbeiten und es weiterzuführen, diese Erfahrung habe ich bis anhin noch nie gemacht.

Einzig und alleine würde ich das nächste Mal früher mit der Integration der Software in das bereits vorhandene Server-System beginnen, da dort immer unvorhergesehene Dinge geschehen können.

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, 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. Da der Rewrite des Import Konverters mehr Zeit in Anspruch nahm als wir eingeplant hatten, übernahm Herr Meier mehr Dokumentationsarbeit.

Mit den Studenten des Vorgängerprojektes haben wir ebenfalls einen guten Draht. Herrn Hof kenne ich von der BMS und er, sowie Mike Huber, gaben uns Hilfestellung wenn wir sie benötigten.

Für die Integration der Diplomarbeit von Herr Busarello war hauptsächlich Herr Meier zuständig. Aber auch Herr Busarello gab uns hilfreichen Input, seine Arbeit in unsere integrieren zu können. Die Integration klappte somit wunderbar.

4.1.2.2 ZEITPLANUNG

In den ersten Semesterwochen war ich zu wenig aktiv am Projekt beteiligt. Die Einarbeitungszeit in das Thema und die Vorgängerarbeit haben wir klar unterschätzt. Hinzu kam noch mein Wohnungswechsel anfangs November. Die etwas verpasste Zeit habe ich jedoch in den letzten Wochen bei Weitem wettgemacht. Der Update-Prozess konnte zwar nicht mehr an das benutzerdefinierte Zielschema der Datenbank angepasst werden, die Gründe hierfür sind jedoch die Grösse und Komplexität der Aufgabe (respektive der OSM-Datenstruktur).

Einige geplante Eigenschaften mussten wir auch in spätem Stadium der Software nochmals überdenken, bzw. sahen erst dann, warum die Konzepte dahinter so nicht umsetzbar sind. Workarounds in der Vorgängerarbeit, welche wir nicht auf Anhieb verstanden hatten, wurden nun klar und wir konnten sie entsprechend anpassen (AreaHandling für single-closed-ways, Mapping Problem bei Relations deren benötigte Tags auf einen Member verlagert wurden).

Leider reichte uns die Zeit nicht mehr, unsere Arbeit auf dem Unix-Zielrechner zum Laufen zu bringen. Der Import der Daten klappte, jedoch wollte Tomcat respektive der GeoServer nicht mitmachen. Dies wollen wir in der kommenden Bachelor-Arbeit natürlich nachholen. Immerhin klappt die Applikation auf einem Windows-Rechner.

4.1.2.3 NEU ERLERNTE TECHNOLOGIEN

- JAXB
- OSM-Daten
- GeoServer

4.1.2.4 FAZIT

Die Arbeit war sehr interessant und es machte auch Spass, sich mit OSM-Kartenmaterial herumzuschlagen. Die Einarbeitungszeit war etwas mühsam, da wir jetzt jedoch im Thema sind, freue ich mich auf die Bachelorarbeit. Es war jedoch eine gute Erfahrung, das Erlernte in einem grösseren Projekt anzuwenden. Wir konnten uns die

Erfahrungen aus dem SE2 Projekt zugute machen und haben weitere dazuerlangt, welche uns in der Bachelorarbeit hilfreich sein werden. Herr Keller begleitete uns stets interessiert und hilfsbereit zum Ziel.

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 2.0/	
	Installationsdateien für OSM-in-a-Box 2.0
/Source/mapcompare	
	Source der Mapcompare Webseite
/Source/osm2gis	
	Source des Import-Konverters osm2gis 2.0. Dieser Ordner kann im Eclipse mit Import an existing Project importiert werden.

4.3 ANHANG B GLOSSAR UND ABKÜRZUNGSVERZEICHNIS

Term	Explanation
CentOS	CentOS 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 Zitier regeln 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 Zitier regeln korrekt angegeben habe.

Ort, Datum:

Name, Unterschrift:

5 DOKUMENT-HISTORY

Da diese Semesterarbeit eine Weiterführung des OpenSteetMap-in-a-Box Projektes ist, wurden gewisse Kapitel dieses Dokumentes aus der Vorgängerarbeit übernommen, abgeändert oder komplett neu geschrieben. Dieses Kapitel gibt Aufschluss auf die Autoren der einzelnen Kapitel.

Die übernommenen Kapitel stammen aus dem Dokument *Gesamtdokumentation.docx Kapitel 5.1*.

5.1 KAPITEL

ABSTRACT	Neu
MANAGEMENT SUMMARY	Neu
1 Aufgabenstellung	Neu
2 Teil I technischer	Übernommen
2.4.1 Volle Konfigurationsmöglichkeiten	Hinzugefügt
2.5 Schlussfolgerungen	Neu
2.6 ausblick	Erweitert
3.1 Anforderungsspezifikation	Wurde von Dokument 03Requirements\Anforderungsspezifikation.docx übernommen. Genaue Dokument-History kann in diesem Dokument nachgeschlagen werden.
3.4 Benutzerdokumentation	Wurde von Dokument 09_Installation_Usermanual\OpenStreetMap-in-a-Box2_Usermanual.docx übernommen. Genaue Dokument-History kann in diesem Dokument nachgeschlagen werden.
0 Projektmanagement	Wurde von Dokument 02_Projektplan/Projektplan.xlsx übernommen. Dieses Dokument wurde neu erstellt.
3.3 Projektmonitoring	Übernommen und angepasst.
4 Anhang	Übernommen und angepasst.
3.2 Design	Wurde von Dokument 05_Design/Softwarearchitecture.docx übernommen. Genaue Dokument-History kann in diesem Dokument nachgeschlagen werden.

5.2 ABBILDUNGEN

Abbildung 5: osm2gis Software Komponenten	Ergänzt