Integrating OpenStreetMap with Public Transport Network Format NeTEx using the JOSM Editor

Projekt Arbeit 2

Labian Gashi

Information and Communication Technologies
Software and Systems

Advisor Prof. Stefan Keller

HSR - Hochschule für Technik Rapperswil

Spring Semester 2020

Abstract

Exchanging complex transport data between different systems is not an easy task. There are a lot of details that need to be taken care of when doing such things, and of course, things like this need some sort of technology standardization after a while, just so that everyone stays on the same page when exchanging such data.

NeTEx (Network Timetable Exchange) is a Technical Standard developed by the European Committee for Standardization (CEN) for exchanging Public Transport schedules and related data among distributed systems. OpenStreetMap (OSM) is a collaborative project for creating a free editable map of the world. It A lot of people and companies contribute to OSM daily to create a realistic map of the world containing the newest additions and the relations between objects in the world.

The idea behind this project is creating a plugin for the Java OpenStreetMap editor (JOSM) that converts OSM data into the NeTEx format, which is represented using XML. JOSM is an open-source advanced OSM editor that offers a lot of advanced features for editing and improving the OSM data and has the ability to integrate different plugins, plugins such as this, in order to add new features to the application or improve the usability of it.

The plugin uses available OSM data and uses different functions to convert it into the NeTEx format. It then tries to find transport data flaws and urge the users to fix them for more accurate future conversions.

Keywords: *NeTEx, CEN, OSM, JOSM, CEN, Map, Plugin, Transport, XML, Java.*

Acknowledgements

Management Summary

Exchanging data between parties is always necessary to have a functioning and robust service. This exchange process however, needs to have some sort of consistency after a while. A standardization technique for such processes must be in place.

The European Committee for Standardization (CEN) is a public standards organization that fosters the economy of the European Union (EU) in global trading and develops various technical standards for various products, materials or processes. Network Timetable Exchange (NeTEx) is a CEN Technical Standard for exchanging public transform data.

In this project, we try to create a plugin for the Java OpenStreetMap editor (JOSM) that converts OpenStreetMap (OSM) data to the NeTEx format and tries to improve the OSM data by suggesting different edits for the OSM elements after the conversion has been finished. The NeTEx data is represented using XML.

Goals

The main goal of this project is creating a JOSM plugin that grabs the currently loaded OSM map data and then does various operations in order to convert such data into the NeTEx format.

The plugin, during the conversion process, tries to find any inconsistencies or problems with the OSM transport data and saves them for the end of the conversion. After the conversion is completed, the plugin takes all the accumulated problems of OSM data and displays them on the JOSM map, suggesting different edits and operations for transport data that need improvement in order to stay consistent with other OSM data and with the usual approach of describing such elements.

This plugin also has to be published under the official JOSM plugin repository and it has to comply with the JOSM development guidelines and rules. The publishing is done for use by the community and maybe for future bug fixes or code improvements.

Results

The plugin has been added to JOSM and has been developed and tested with real OSM data. There exists a lot of inconsistent data within JOSM, the plugin tries to fix/map such data itself during the conversion by putting different conditions, and if that data cannot be mapped or fixed by the algorithm, it means that an OSM element is missing some sort of tag that is used for identification. These elements with missing/wrong tags are displayed at the end of the conversion.

The plugin first takes the OSM data and creates NeTEx Java classes that have been generated by an XML binding framework, which is called JAXB and then marshalls them at the end and writes them in an XML-format file in order to complete the conversion and the export of the OSM data into NeTEx.
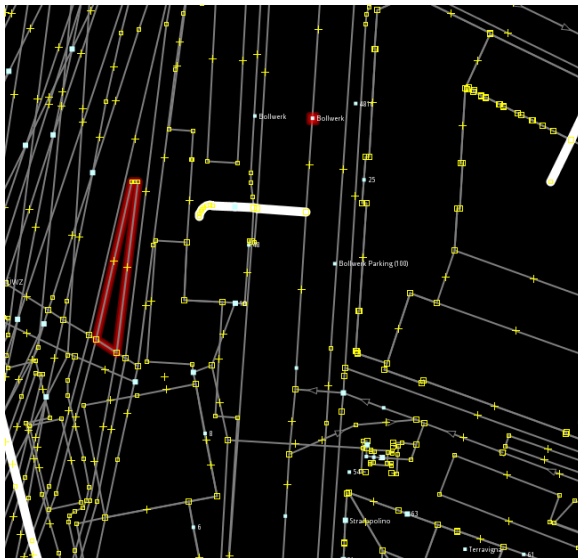

FIGURE 1: A look at how OSM elements and relations are displayed in JOSM

FIGURE 2: A snippet of some exported NeTEx data which was converted from OSM data using this plugin

Future Outlook

The current version of this plugin successfully converts OSM data into NeTEx with a single click. Future versions can be improved in order to take care of even more conditions and data inconsistencies, improve the OSM data by providing even more suggestions etc.

Another very nice/important thing to add to this plugin would be implementing an algorithm for finding the associated transport objects within OSM. Some objects (like footpaths or steps) exist in or near stop places just for the purpose of leading to a platform, station etc. Such objects, don't have any tags that display what station or platform they serve. An algorithm/approach that finds such elements and their relations would be something that would improve the plugin's main algorithm by far.

New features can be added in the future to the plugin that would consider even more transport data (such as ferries, airports, metro stations etc) and even more NeTEx transport-related elements.

Declaration of Authorship

I hereby declare that:

- This thesis and the work reported here was composed by and originated entirely from me unless stated otherwise in the assignment of tasks or agreed otherwise in writing with the supervisor.

- All information derived from the published and unpublished work of others has been acknowledged in the text and references are correctly given in the bibliography.

- No copyrighted material has been used illicitly in this work.

Place, Date   Signature

Contents

Chapter 1

Introduction

The documentation for this project is split into four main parts. The first chapter gives an introduction to the goals and requirements that this project was initially set to meet. It also explains some of the important technologies & tools, keywords and notations that are crucial to this project.

The second chapter has to do everything with the architecture and the design of the application. It explains about the technologies that were used, the reason why they were used and the benefits of using those technologies for this project.

After the things above have been clarified, the third chapter explains about the actual technical implementation aspects of the plugin. It mostly explains about some of the important methods/approaches that were used to achieve the goal of this plugin and the reason why those methods were decided to be used. It also explains about the main technical notations about the project, what they mean and how they were used in the benefit of this project work.

The fourth and final chapter presents the results that were achieved from this project work and the outlook of it. It also wraps up with a conclusion about the project and my personal reflection on this whole project work.

1.1   Requirements

The main objective for this project is creating an OpenStreetMap (OSM) plugin for the Java OpenStreetMap Editor (JOSM) editor that converts the OSM data into an XML format called NeTEx, which stands for Network Timetable Exchange and is a CEN technical standard for exchanging public transport information.

The plugin takes existing OSM data, converts it into NeTEx and then logs important information such as errors, warnings regarding the OSM transport data and the NeTEx conversion into the JOSM application map layer.

The main tasks include (priority sorted):

- Creating a JOSM friendly plugin that converts OSM data into the NeTEx format.

- Improving further NeTEx conversions by suggesting different OSM edits that benefit the NeTEx conversion.

- Incorporating the NeTEx conversion and the improvement suggestions with the JOSM default workflow.

The plugin sticks to the official NeTEx and OSM rules throughout its whole creation and workflow. It is also built under the official JOSM development guidelines and it maximally utilizes the native JOSM methods and libraries that are key to this plugin.

It is licensed under the GPL license.

1.2 **OpenStreetMap - OSM**

OpenStreetMap (OSM) is a project to build a free geographic database of the world. Its aim is to eventually have a record of every single geographic feature on the planet. While this started with mapping streets, it has already gone far beyond that to include footpaths, buildings, waterways, pipelines, woodland, beaches, postboxes, and even individual trees. Along with physical geography, the project also includes administrative boundaries, details of land use, bus routes, and other abstract ideas that aren't apparent from the landscape itself. [1]

OpenStreetMap is powered by open-source software such as editing software, various APIs etc. One can extract very sophisticated information from the geographical data the OSM consists of. Various users (end-users, developers, maintainers etc.) all around the world contribute daily to improving the geographical data in OSM and to improving the ease of use of such data and information. The OSM data can be used in multiple ways such as producing paper and electronic maps, integrating such data into your own applications and route planning.

There are already a lot of famous users that utilize OSM such as: Facebook, Craigslist, Seznam etc. OSM is *community-driven* and the community of contributors mainly consists of enthusiast mappers, GIS professionals, humanitarians and many more. It is also *open-data*, which means that anyone can freely use OSM for any purpose as long as OSM and its contributors are credited. The OSM map data can be used for web sites, mobile apps and various hardware devices.

1.3   Java OpenStreetMap Editor - JOSM

Java OpenStreetMap Editor (JOSM) is an extensible editor for OSM for Java [2]. It is open-source and developer using the Java programming language. It is a desktop application that offers a lot of options and is used for editing OSM data and their meta-data tags. It has a steep learning curve and may look complex at first sight, but it's very popular because of its continuous contributions, its plugins and its stability. Eventhough OSM has many editors, JOSM is considered to be one of the most advanced and professional ones (hence the steep learning curve). It is maintainted under the GNU General Public License v3.0.

JOSM Plugins extend or modify the basic features of the JOSM editor. They are created from enthusiastic developers all around the world, under the JOSM application. There is a repository of plugins maintained in JOSM where users can pick and choose which plugin they want to add to their JOSM editor. If some plugin is not available in the repository (which it should be), it can be installed manually pretty easily.

1.4   Network Timetable Exchange - NeTEx

Network Timetable Exchange (NeTEx) is a CEN Technical Standard for exchanging Public Transport schedules and related data. It provides a means to exchange data for passenger information such as stops, routes timetables and fares, among different computer systems, together with related operational data. It can be used to collect and integrate date from many different stakeholders, and to reintegrate it as it evolves through successive versions. [3]

It is divided into three parts:

- **Part 1 - Network Topology**: Describes the Public Transport Network topology.

- **Part 2 - Timing Information**: Describes Scheduled Timetables.

- **Part 3 - Public Transport Fares**: Covers Fare information.

NeTEx is intended to provide a European wide standard for exchanging Public Transport data for Passenger Information. It is based on the CEN Transmodel which is the CEN European Reference data model for public transport.

Chapter 2

Architecture and Design

This chapter describes a little bit about the overview of the plugin, the programming language that was used, how the plugin is deployed, what it needs in order to be executed successfully etc. It also describes why that specific programming language and the libraries were chosen and the main reasons sitting behind those decisions. The Plugin section of this chapter describes the plugin in a more-depth manner, some of the libraries that were used, how it is bundled together and the components that are crucial to the plugin.

2.1   Overview

The plugin is developed and built using the Java programming language (Java Platform SE 8), which is fully compliant with the JOSM development environment (JOSM also supports higher Java versions). The plugin is developed as a "separate" component from JOSM, which can be manually added/installed to an existing JOSM application environment.

The plugin is ultimately deployed as a single *.jar* file, which contains all the necessary components, dependencies and libraries that the plugin needs in order to execute successfully. This jar file however, does not serve its purpose if executed solely, it requires a JOSM application environment and must be executed from it in order for it to serve its purpose. It is a relatively heavy .jar file because of the XML components that it has bundled within, which boil down to a Java Model of the whole NeTEx XML schema containing all its components, their necessary interrelations and methods.

2.2   Design Decision - Programming Language

This plugin was built using the Java programming language, version 8.

Java is a powerful general-purpose programming language. It is used to develop desktop and mobile applications, big data processing, embedded systems, and so on. According to Oracle, the company that owns Java, Java runs on 3 billion devices worldwide, which makes Java one of the most popular programming languages. [4]

The reason why this plugin was chosen to be built and developed using Java is because of JOSM, it runs on Java, so everything under it must run on Java too, including the plugins. All of the JOSM plugins are developed inside the JOSM environment as separate .jar files and must use JOSM native Java methods in order to be executed and incorporated within JOSM.

It is also an Object-Oriented programming language, which runs under ANY operating system (because of it's compiler) and uses inheritance and abstract methods (Object Oriented programming principles), which turned out to be very useful and neat for the nature of this plugin.

Here's a snippet of Java code & syntax example of a Java Program that computes the quotient and the remainder of a division: [5]

```java
public class QuotientRemainder {

    public static void main(String[] args) {

        int dividend = 25, divisor = 4;

        int quotient = dividend / divisor;
        int remainder = dividend % divisor;

        System.out.println("Quotient = " + quotient);
        System.out.println("Remainder = " + remainder);
    }
}
```

And the result from running the program would be:

```
Quotient = 6
Remainder = 1
```

2.3 The Plugin

JOSM Plugins extend or modify the feature set of the JOSM editor. [6]
That is the main purpose of these plugins. They are created by the JOSM community
of developers and they tend to improve JOSM usability by adding extra or "missing"
functionality, or in the most cases, by adding a completely new feature that interacts
with JOSM components but serves an exclusive purpose.
There exist plugins for improving the look & feel of drawing different shapes of objects
within JOSM, for converting OSM data into various formats (which is our case), for ex-
porting OSM data into formats such as PDF or GPX, for showing additional important
information to some users etc.

This plugin is a single executable .jar file which runs under JOSM and is executed
within it. It is developed by following the official JOSM development guidelines and
OSM rules. The plugin, if installed on the JOSM environment, is first executed using a
JOSM native library which takes care of plugin initialization, execution and error han-
dling. After loaded into JOSM, depending on the nature of the plugin, it can interact
with any component within JOSM, in this case, the plugin is a single option on the
toolbar menu that simply takes the currently loaded map data in the main map layer
and converts it into the NeTEx format.
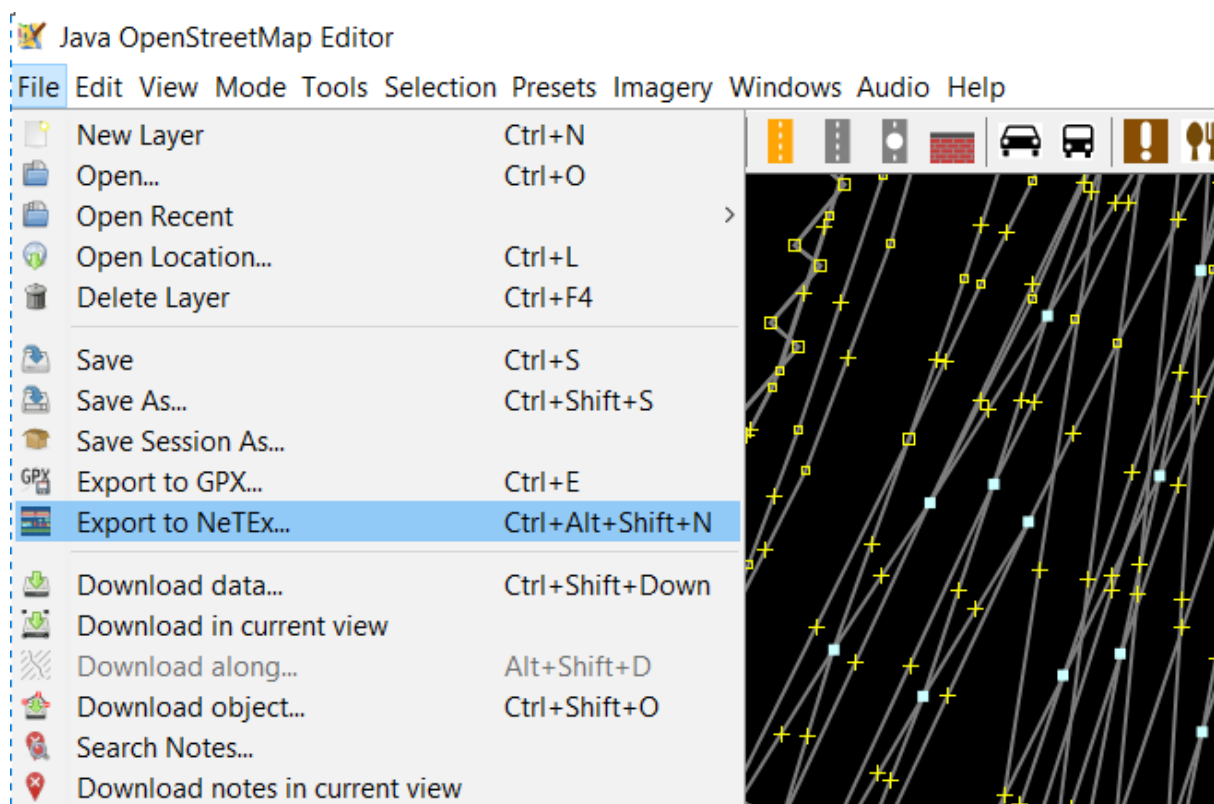
FIGURE 2.1: The location of the NeTEx Converter plugin menu item in the JOSM toolbar

After the conversion is completed, any log information (warning, error) will be displayed within the JOSM map layer by highlighting the intended object, may that be a node, a way or even a relation.
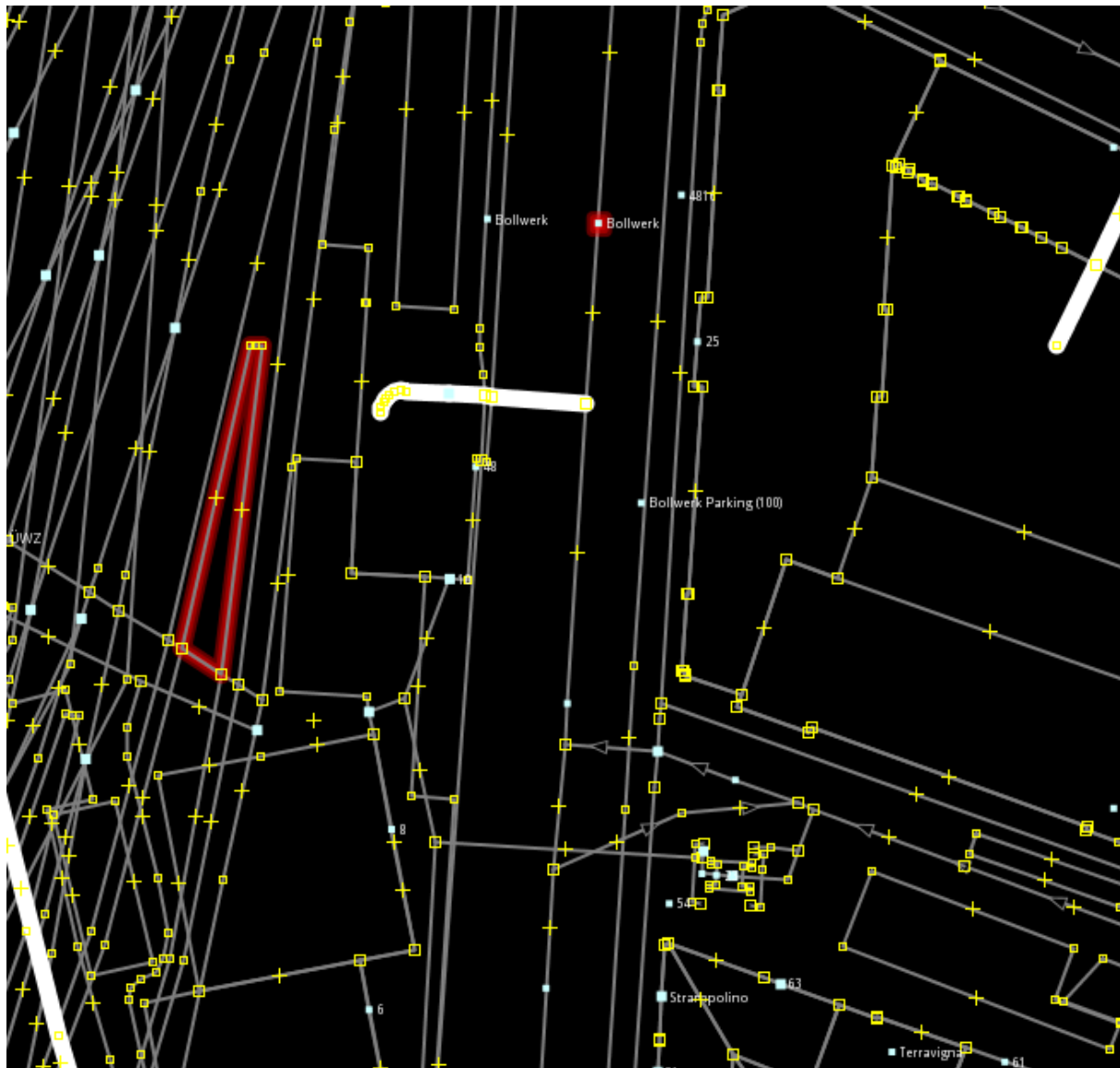
FIGURE 2.2: An example of the plugin highlighting objects that need attention after a conversion (Location of the map layer - Bern, Switzerland)

After identifying the highlighted objects, if we click on them, on the tags of the object, there will be a tag with a message corresponding to the action/s that need/s to be taken in order to improve that object so that the future NeTEx conversions are refined. All of the tags generated by the plugin that need attention have a suffix of " - *(NeTeX Converter)*". If after the conversion, there are a lot of highlighted objects in the JOSM map layer, it means that the OSM data is not very suitable for a informative NeTEx conversion. When this happens, the user should improve the tags that are suggested by the plugin, correcting them or adding them completely as new in order for them to

be fully compliant and convertible by the plugin. After the corrections, the next NeTEx conversions will be much more improved and informative.

```
7009    <StopPlace id="ch:1:StopPlace:8509000">
7010        <Name>Chur</Name>
7011        <PrivateCode>org:osm:node:2522151337</PrivateCode>
7012        <Centroid>
7013            <Location>
7014                <Longitude>9.5296459</Longitude>
7015                <Latitude>46.8535563</Latitude>
7016                <Altitude>585</Altitude>
7017            </Location>
7018        </Centroid>
7019        <AccessibilityAssessment>
7026        <equipmentPlaces>
7208        <PublicCode>8509000</PublicCode>
7209        <StopPlaceType>railStation</StopPlaceType>
7210        <quays>
7525        <pathLinks>
8649        <pathJunctions>
9497    </StopPlace>
```

FIGURE 2.3: A snippet of an exported NeTEx document from the plugin (Location of the exported data - Chur, Switzerland)

Chapter 3

Implementation

3.1 Overview

This part of the documentation describes how the project was implemented in more-detailed and technical manner. It shows some of the key approaches that were used to solve some of the major conversion obstacles. It explains what kind of data OSM contains and how that data was used in conjunction with the plugin to produce an XML document compliant with the NeTEx schema. As we know, OSM data is not very consistent, since it's updated by the community, which means that a lot of extra measures and conditions had to be taken in order to avoid and solve problems of data inconsistency.

This section will also cover some of the important techniques and tools that were used to test, document the plugin, generate artifacts from the plugin etc. It will display the code repository features and the issue tracking tools, which for this application were hosted/maintained on project called GitLab, which is hosted under an open-source license. Lastly, it will show the document preparation system that was used to create this very document.

3.2　OpenStreetMap Data

Elements (or objects) are the basic components of OpenStreetMap's conceptual data model of the physical world. They consist of:

- Nodes (defining points in space)

- Ways (defining linear features and area boundaries)

- Relations (which are sometimes used to explain how other elements work together).

All of the above can have one or more associated tags (which describe the meaning of a particular element). [7]

3.2.1　Data Types

3.2.1.1　Node

A node represents a specific point on the earth's surface defined by its latitude and longitude. Each node comprises at least an id number and a pair of coordinates.
Nodes can be used to define standalone point features. For example, a node could represent a park bench or a water well.
Nodes are also used to define the shape of a way. When used as points along ways, nodes usually have no tags, though some of them could. For example, *highway=traffic_signals* marks traffic signals on a road, and *power=tower* represents a pylon along an electric power line.
A node can be included as member of relation. The relation also may indicate the member's role: that is, the node's function in this particular set of related data elements. [7]

3.2.1.2　Way

A way is an ordered list of between 2 and 2,000 nodes that define a polyline. Ways are used to represent linear features such as rivers and roads.
Ways can also represent the boundaries of areas (solid polygons) such as buildings or forests. In this case, the way's first and last node will be the same. This is called a "closed way".
Note that closed ways occasionally represent loops, such as roundabouts on highways, rather than solid areas. The way's tags must be examined to discover which it is.
Areas with holes, or with boundaries of more than 2,000 nodes, cannot be represented by a single way. Instead, the feature will require a more complex multipolygon relation data structure. [7]

3.2.1.3   Relation

A relation is a multi-purpose data structure that documents a relationship between two or more data elements (nodes, ways, and/or other relations). Examples include:

- A route relation, which lists the ways that form a major (numbered) highway, a cycle route, or a bus route.

- A turn restriction that says you can't turn from one way into another way.

- A multipolygon that describes an area (whose boundary is the 'outer way') with holes (the 'inner ways').

Thus, relations can have different meanings. The relation's meaning is defined by its tags. Typically, the relation will have a 'type' tag. The relation's other tags need to be interpreted in light of the type tag.
The relation is primarily an ordered list of nodes, ways, or other relations. These objects are known as the relation's members.
Each element can optionally have a role within the relation. For example, a turn restriction would have members with "from" and "to" roles, describing the particular turn that is forbidden.
A single element such as a particular way may appear multiple times in a relation. [7]

3.2.2   Usage of OpenStreetMap Data

All the OpenStreetMap data types explained in the Data Types can represent various transport-related objects. These data types can represent various transport-related information for the plugin:

- A node can represent a bus station, a bus stop, a train station, elevators near train stations etc.

- A way (be that closed or open ways), can represent footpaths near stations, various steps or ramps that are located in the train station that lead to train platforms, bus platforms etc. They can also represent bus platforms or even train platforms in some cases.

- A relation has children, those children can be of any type, they can even be relations themselves. Relations represent train platforms, bus routes, train routes and require special care from the plugin in order to serve their purpose.

All of this OSM data is contained in the JOSM editor and the plugin has access to JOSM interfaces in order to manipulate such data, which is of course, necessary for

the conversion.

The plugin initially checks every visible element within the JOSM map layer and checks their tags. Their tags then represent what kind of an element that is and if it is relative to our plugin. After that, the plugin finds the relative elements and their type, and then depending on the type, different NeTEx objects are created. While the NeTEx objects are being created, the conversion algorithm adds various attributes to those objects depending on the tags that they contain, what they are close to and their importance to the related transport information. Different elements that are found at later stages can be related to previously identified relative elements, when that happens, the algorithm finds the connection of those elements and relates/fixes them accordingly.

3.3   Java OpenStreetMap Editor

Java OpenStreetMap Editor is an open-source desktop application written in Java that is used in order to edit OSM content in a more advanced/professional manner. The source code of JOSM is hosted in the Apache Subversion (SVN) version control system and is compiled and built using the Apache Ant Java library and command-line tool.

First, the project must be cloned in the target machine using SVN or Git and then must be compiled using Ant, which will create a .jar file, containing the JOSM application, hosted locally on your machine. The JOSM project has a directory called *plugins* which contains all of the available JOSM plugins. Every plugin contains a *build.xml* file which contains the common Ant building information, plus some mandatory fields for the plugin, such as: The plugins version, the entry point (main class) of the plugin, its minimum Java verison, the plugin icon etc. If you want to register and publish a plugin in the official JOSM plugin repository, a set of procedures must be followed before that is possible. The benefit of that is that the plugin will be available to the whole JOSM community and other developers might fix bugs or improve the code.

JOSM has a plugin API that must be imported as a package from your plugins main class and then there are specific methods and entry points that the plugin must implement in order for JOSM to take the necessary steps into initializing the plugin. These methods can specify anything, some of them are required and some of them are optional. With the help of this API, you can set the plugins location, its icon, its name, whether its disabled/enabled initially etc.
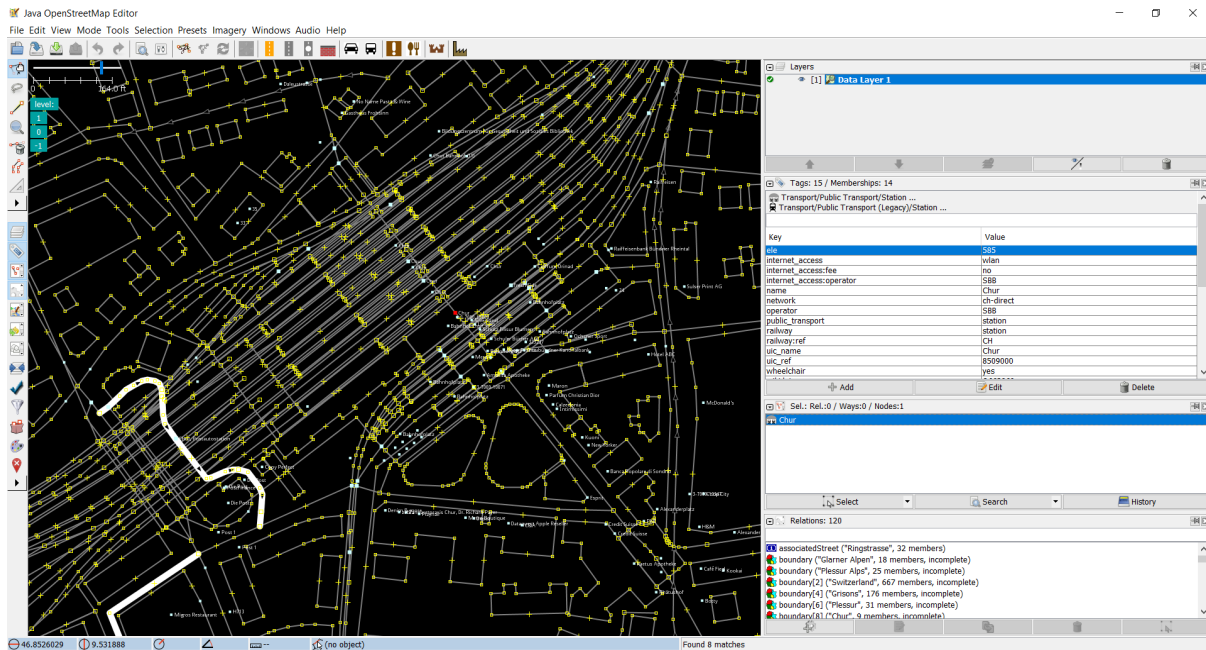
FIGURE 3.1: A taste of the JOSM editor

3.4 XML Binding Framework

NeTEx is a general purpose XML format designed for the efficient exchange of complex transport data among distributed systems. In order to write such XML in a programmatic manner, there must exist a model which is written in the intended language that represents the components of such format, their attributes and their interrelations. In this case, since the plugin is written in Java, a Java model of the NeTEx schema is required for coding. To achieve such a thing, the Jakarta XML Binding (JAXB) framework was used.

Java Architecture for XML Binding (JAXB) provides a fast and convenient way to bind XML schemas and Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications. As part of this process, JAXB provides methods for unmarshalling (reading) XML instance documents into Java content trees, and then marshalling (writing) Java content trees back into XML instance documents. JAXB also provides a way to generate XML schema from Java objects . [8]

It simplifies access to an XML document from a Java program by presenting the XML document to the program in a Java format.

3.4.1 The Binding Procedure

The official NeTEx schema is taken from the official NeTEx GitHub Repository using a batch file which downloads the schema (.xsd file) and using the JAXB files and configuration, creates the corresponding Java classes for each NeTEx object and it's relations.

After this whole set of classes are generated (which are put into a Java package), they are bundled together with the plugin code in order to generate a single .jar file at the end, which is of course, executable using the JOSM editor.

The framework generates a lot of Java classes and sub-packages that lead to pretty heavy content, which implies that the ultimate .jar file is not a light-weight .jar file but rather a relatively heavy one (10MB). This package of generated classes is then imported by the plugins main conversion classes and used accordingly in order to enable the conversion process. After the conversion is completed, the whole NeTEx content is then "marshalled" into XML content and written in an XML file that is compliant with the official NeTEx schema.

As of now, there are a total of 2,704 generated files from JAXB.

Some of the generated classes from JAXB (for the Stop Place):

FIGURE 3.2: Some of the JAXB generated NeTEx model classes (Stop Place)

3.5   NeTEx Conversion

The main technical approaches and key points used to convert OSM data into NeTEx are described in this section.

The plugin initially processes and reads all of the loaded elements in the JOSM map layer (Nodes, Ways, Relations) and carefully checks if any element is relative to any NeTEx object. When that happens, the plugin creates a NeTEx object from it using a separate "parser" class (but not all of the relative elements initially). The algorithm for converting the data into NeTEx is contained in the *NeTExExport.java* class, which is the class containing the heavy logic of the conversion process.
This class contains logic such as:

- Finding the relative OSM elements for NeTEx.

- Calling the appropriate methods for parsing such objects into NeTEx.

- Logging different relative information during the NeTEx conversion.

- Accumulating them so that at the end of the conversion, they are appended to the JOSM objects, which are then highlighted after the conversion is completed.

First, the plugin identifies only the OSM elements that correspond to NeTEx *StopPlace* objects. After all of these are created (which are the top parent nodes), then we re-iterate through the OSM elements, trying to find other important elements such as: platforms, elevators, footpaths, steps etc. A lot of ambiguities and discrepancies are resolved while these loops are being executed. All the relevant information and children of a *StopPlace* are mapped and resolved here, accumulating them all together and then assigning them to their corresponding *StopPlace* using the generated NeTEx classes from the JAXB framework.

Portion of code loop that iterates through all of the visible OSM elements and tries to identify the relevant elements:

```java
for (OsmPrimitive primitive : primitives) {
  if (primitive instanceof Node) {
  Node node = (Node) primitive;

  if (OSMHelper.isTrainStation(node)) {
    stopPlaces.put(node, neTExParser.createStopPlace(node,
    ↪  StopTypeEnumeration.RAIL_STATION));
  }
  else if (OSMHelper.isBusStation(node)) {
    stopPlaces.put(node, neTExParser.createStopPlace(node,
    ↪  StopTypeEnumeration.BUS_STATION));
  }
  else if (OSMHelper.isBusStop(node)) {
    stopPlaces.put(node, neTExParser.createStopPlace(node,
    ↪  StopTypeEnumeration.ONSTREET_BUS));
  }
  else if (OSMHelper.isPlatform(node)) {
    quays.put(node, neTExParser.createQuay(node));
  }
  else if (OSMHelper.isElevator(node)) {
    elevators.put(node, neTExParser.createElevator(node));
  }
...
}
```

A lot of conditions are available in the algorithm because of the OSM data inconsistency. The plugin tries to "solve" or "mitigate" such inconsistency by providing log messages at the end of the conversion, which, if fixed, make the data more consistent in general and benefit the NeTEx conversion a lot. Some warnings are more critical than the others.

Portion of code extracted that shows the generation of the accumulated warning messages:

```java
for (PrimitiveLogMessage logMessage : LOG_MESSAGES) {

  OsmPrimitive primitive =
  ↪   ds.getPrimitiveById(logMessage.getPrimitiveId(),
  ↪   logMessage.getPrimitiveType());

  if (primitive != null) {
    Logging.warn(tr("Warnings found for the {0} with the id: {1}",
      primitive.getType(), Long.toString(primitive.getId())));

    primitive.setHighlighted(true);

    for (Map.Entry<String, String> entry :
    ↪   logMessage.getKeys().entrySet()) {
      primitive.put(entry.getKey(), entry.getValue());
    }
  }
}
```

3.6 Testing

This plugin has been unit tested with the JUnit testing framework.
The unit testing that was done here mainly consisted of testing the JAXB marshaller, the NeTEx java model classes, some XML validations etc.

3.6.1 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use. [9]
The Unit Testing for this project has been done using the JUnit testing framework. JUnit is a framework to write repeatable unit tests. It is an instance of the xUnit architecture. It is one of the most popular unit testing frameworks for Java.

The unit testing for this plugin was mainly done to test features such as:

- Marshalling and unmarshalling of Java objects into XML (for NeTEx)

- Validating the XML content at the end of the conversion

- Creating a complex dummy train station and testing the Java classes generated from JAXB

- Writing XML content into a file etc.

The unit testing classes are stored in a separate package (possibly complete folder separation in IDE) and have some other commands compared to the normal Java packages. It is very important and efficient to let the IDE know what your test folder is and what your main folder is, so that the IDE knows how to treat them accordingly.

```
init:
Deleting: C:\Users\labia\Desktop\hsr\semester_2\courses\Z_ProjektArbeit_2\josm\josm\plugins\netex_converter\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\labia\Desktop\hsr\semester_2\courses\Z_ProjektArbeit_2\josm\josm\plugins\netex_converter\build\built-jar.properties
compile:
compile-test:
Testsuite: unit.org.openstreetmap.josm.plugins.netex_converter.netex.ComplexStationMarshallerTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.237 sec

Testsuite: unit.org.openstreetmap.josm.plugins.netex_converter.netex.NeTExValidatorTest
[main] INFO com.netex.validation.NeTExValidator - Loading resource: xsd/NeTEx-latest/NeTEx_publication.xsd
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.299 sec

------------- Standard Error ----------------
[main] INFO com.netex.validation.NeTExValidator - Loading resource: xsd/NeTEx-latest/NeTEx_publication.xsd
------------- ---------------- ---------------
test-report:
test:
BUILD SUCCESSFUL (total time: 9 seconds)
```

FIGURE 3.3: An example of results after running some JUnit tests

3.7   Code Repository and Software Lifecycle

The code repository and software lifecycle features for this project were provided by GitLab.

GitLab is a DevOps lifecycle tool that provides a lot of things like: Git-repository manager providing wiki, issue-tracking and CI/CD pipeline features, it uses an open-source license and is developed by GitLab Inc. It has a very nice web interface and lots of different features that make application lifecycle much easier. The features GitLab offers that were used for this project were the code repository host, issue tracking tools etc.
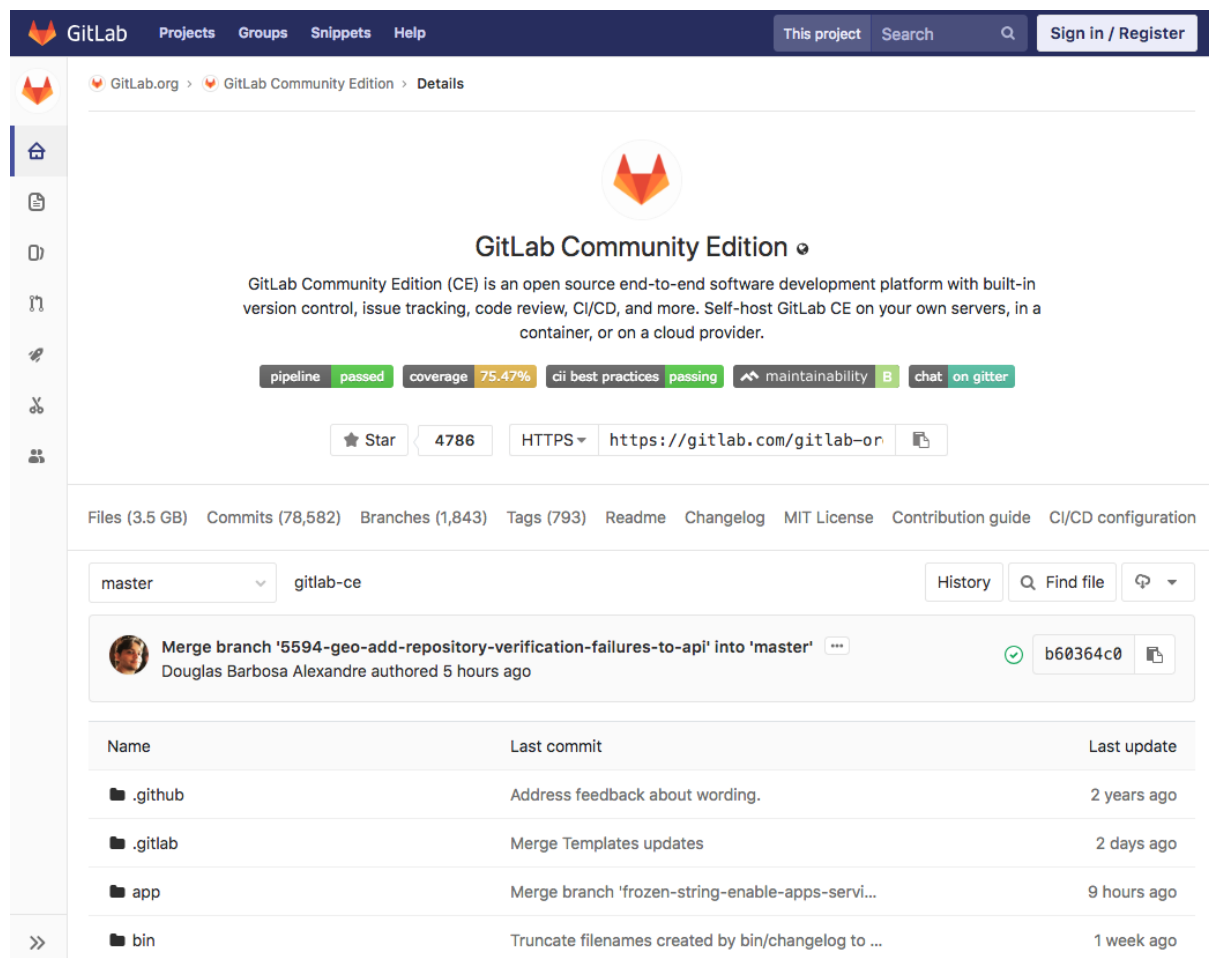

FIGURE 3.4: A sneak peek of the GitLab interface

3.8 Document Preparation System

The typesetting system that was used in order to document this project was LaTeX. LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents. LaTeX is available as free software. [10]
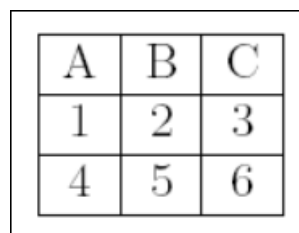
The application that was used in conjunction with LaTeX was TeXstudio. TeXstudio is an integrated writing environment for creating LaTeX documents. Our goal is to make writing LaTeX as easy and comfortable as possible. Therefore TeXstudio has numerous features like syntax-highlighting, integrated viewer, reference checking and various assistants. [11]

The reason why LaTeX was used is that it is very convenient to developers, since they can get used to the syntax very quickly and are familiar with "coding" approaches. Another big reason is that it provides a lot of packages for displaying code in a nice formatted way, it has the ability of displaying web links in a nice manner and also the figures and figure alignments are very appropriate and easy to create & maintain. A taste of LaTEx: [12]

```latex
\documentclass[12pt,twoside,a4paper]{article}
\begin{document}
\begin{tabular}{|c|c|c|}
\hline
A & B & C \\
\hline
1 & 2 & 3 \\
\hline
4 & 5 & 6
\\
\hline
\end{tabular}
\end{document}
```

This would produce the following output:

FIGURE 3.5: LaTEx document output example [12]

Chapter 4

Results

4.1 Achievements

The main goal of this project described below has been met. This goal contained a lot of sub-goals which ultimately dwell down to one thing: converting OSM data into the NeTEx format.

Integrating OpenStreetMap with Public Transport Network Format "NeTEx" using the JOSM editor:

- Create a JOSM plugin in Java for converting OSM data into NeTEx.

    - Create a JOSM friendly plugin.

    - Convert and export the loaded OSM data of JOSM into a single .XML file using the generated NeTEx Java model.

    - Fix data inconsistency problems by identifying and saving missing tags/information

    - Log the warning messages created while converting the data in the JOSM map layer relative elements

    - Have the plugin deployed in a single *.jar* file.

- Translate the NeTEx XML schema into a Java model.

- Publish the plugin in the official JOSM plugins repository.

4.2   Reflection

I have never used OpenStreetMap before, let alone JOSM. This project made me learn a lot of key factors that drive OSM and make OSM such a successful project for the community. It also improved my ability to work in an open-source project and know the approaches that are used by developers when developing open-source projects.

I wasn't very informed about transport data either and never heard of NeTEx before either, so a combination of OSM and NeTEx was very beneficial for me. It is also the first time I have created something that is similar to converting data of a format into a completely different format. The process was a tough challenge but also a very rewarding one.

I have had a lot of experience with working with Java before, but not that much experience of coding in open-source projects where you have to code by following "protocol". I had to do a lot of research within the code, but after a while, I understood how the whole thing worked and then after that, everything in the code made sense and was straight-forward to reach in the code.

I have also never worked with an XML binding framework either and it was wonderful to see how such things are configured and implemented, without such frameworks, this would be impossible! A lot of technologies and approaches used here were a new experience for me and even though it was overwhelming sometimes, it was definitely worth it!

List of Figures

Appendix A

Installation

There are two ways to install this plugin. Install it as a plugin into JOSM or install it in a development environment for developing further features, bug fixes or code refactoring.

To install it as a plugin for your JOSM environment (major cases), there are two approaches:

- Installing them from within JOSM (when they are officially published plugins) or

- Manually installing them

The below portion of text has been taken directly from the JOSM plugins wiki page instructions: [6]

Usual, easy method
You can easily install plugins from within JOSM as follows

1. Start JOSM, open the preferences window (Edit $\rightarrow$ Preferences or use the toolbar icon) and select the plugins tab.

2. Click on "Download List" to download the list of available plugins.

3. Check the plugins you want installed.

4. Click the update button. All new plugins should start downloading and installing.

5. Restart JOSM. (Some Plugins work even without a restart. You will be notified, if a restart is required.)

Manually install JOSM plugins

Usually you don't need to install plugins manually! If you do need to install manually:

- Save plugin jar files in the plugins directory in the JOSM settings directory:

  - `~/.josm/plugins`, or `Users/<YourName>/AppData/Roaming/JOSM/plugins` for Windows.

  - `/home/\$USER/.josm/plugins` for Linux, or

  - `Users/<YourName>/Library/JOSM/plugins` on MacOS.

A.0.1 Building the .jar file

In order to build the .jar file, the repository of the plugin must first be cloned into your local machine. After that, when in the repository, the following commands need to be run:

```
ant clean
```

```
ant dist
```

This will build the plugin and generate a single .jar file at the end.

This .jar file then needs to be added to the directory mentioned above (how to manually install JOSM plugins). After that, the plugin will be available at the preferences window and ready to be activated.

Appendix B

NeTEx Data

Here are some samples of NeTEx data, displayed using code highlighting (in XML).

B.1   Examples of NeTEx Data

B.1.1   Base Schema Example (No Stop Places)

```
1  <PublicationDelivery xmlns="http://www.netex.org.uk/netex"
   ↪  xmlns:ns2="http://www.opengis.net/gml/3.2"
   ↪  xmlns:ns3="http://www.siri.org.uk/siri">
2    <PublicationTimestamp>1900-01-01T00:00:01</PublicationTimestamp>
3    <ParticipantRef>participantRef</ParticipantRef>
4    <Description lang="en">OSM to NeTEx</Description>
5    <dataObjects>
6      <CompositeFrame id="foo:1:CompositeFrame">
7        <frames>
8          <ResourceFrame id="foo:1:ResourceFrame" />
9          <SiteFrame id="foo:1:SiteFrame">
10           <stopPlaces>
11           </stopPlaces>
12         </SiteFrame>
13       </frames>
14     </CompositeFrame>
15   </dataObjects>
16 </PublicationDelivery>
```

B.1.2 Stop Place Example (StopPlace)

The following Stop Place is taken from a bus stop in Chur, Switzerland:

```
1  <StopPlace id="ch:1:StopPlace:8581896">
2    <Name>Giacomettistrasse</Name>
3    <PrivateCode>org:osm:node:984713668</PrivateCode>
4    <Centroid>
5      <Location>
6        <Longitude>9.5299602</Longitude>
7        <Latitude>46.8638106</Latitude>
8      </Location>
9    </Centroid>
10   <AccessibilityAssessment>
11     <limitations>
12       <AccessibilityLimitation>
13         <WheelchairAccess>false</WheelchairAccess>
14       </AccessibilityLimitation>
15     </limitations>
16   </AccessibilityAssessment>
17   <PublicCode>8581896</PublicCode>
18   <StopPlaceType>onstreetBus</StopPlaceType>
19   <quays>
20     <Quay id="ch:1:Quay:8581896:984713668">
21       <PrivateCode>org:osm:node:984713668</PrivateCode>
22       <Centroid>
23         <Location>
24           <Longitude>9.5299602</Longitude>
25           <Latitude>46.8638106</Latitude>
26         </Location>
27       </Centroid>
28       <PublicCode>984713668</PublicCode>
29       <QuayType>busStop</QuayType>
30     </Quay>
31   </quays>
32 </StopPlace>
```

B.1.3 Platforms Example (Quay)

The following bus platforms are taken from a bus stop in Chur, Switzerland:

```
1  <quays>
2    <Quay id="ch:1:Quay:8594421:6106925323">
3      <PrivateCode>org:osm:node:6106925323</PrivateCode>
4      <Centroid>
5        <Location>
6          <Longitude>9.5163354</Longitude>
7          <Latitude>46.8521078</Latitude>
8        </Location>
9      </Centroid>
10     <PublicCode>6106925323</PublicCode>
11     <QuayType>busStop</QuayType>
12   </Quay>
13   <Quay id="ch:1:Quay:8594421:2446137127">
14     <PrivateCode>org:osm:node:2446137127</PrivateCode>
15     <Centroid>
16       <Location>
17         <Longitude>9.5164318</Longitude>
18         <Latitude>46.8519792</Latitude>
19       </Location>
20     </Centroid>
21     <PublicCode>2446137127</PublicCode>
22     <QuayType>busStop</QuayType>
23   </Quay>
24 </quays>
```

B.1.4 Stairs Example (SitePathLink)

The following stairs are taken from a bus station in Chur, Switzerland:

```xml
1   <SitePathLink id="ch:1:SitePathLink:464925429">
2     <From>
3       <PlaceRef ref="ch:1:PathJunction:4778556657" />
4     </From>
5     <To>
6       <PlaceRef ref="ch:1:PathJunction:4599739661" />
7     </To>
8     <AccessibilityAssessment>
9       <limitations>
10        <AccessibilityLimitation>
11          <WheelchairAccess>false</WheelchairAccess>
12          <StepFreeAccess>false</StepFreeAccess>
13        </AccessibilityLimitation>
14      </limitations>
15    </AccessibilityAssessment>
16    <AccessFeatureType>stairs</AccessFeatureType>
17    <equipmentPlaces>
18      <EquipmentPlaceRef ref="ch:1:EquipmentPlace:464925429" />
19    </equipmentPlaces>
20  </SitePathLink>
```

Appendix C

Project Management

C.1   Organization

The work for this project has been done mainly by me (coding completely by me), with the help of of my supervisor: Prof. Stefan Keller and with a lot of technical help regarding OSM and transport data from Adrian Aeschbacher, he was very helpful and gave me a lot of important information for transport data and OSM. He also tested a lot of the exported NeTEx documents from the early stages of the coding part and up until the end.

The work done was coordinated and supervised also by some of the other SBB staff members, where we held a "stand-up" call every other week in order to find out what was done, the obstacles and the further plans.
The plugin is open source and licensed under GPL.
The source code for this project (plugin) can be found here:
https://gitlab.com/labiangashi/josm-plugin-netex-converter

C.2 Planning and Coordination

Only the first stand-up, which was more of a project kick-off meeting was held physically with me and my supervisor in Rapperswil, in the HSR institute to discuss the initial steps and requirements for the project.

After the first stand-up, online conferences were held which consisted of me, my supervisor, Raphael Das Gupta and some of the SBB staff members responsible/involved in this project. The meetings were held online because of distance but also because of the COVID-19 pandemic.

An agile software development process was used and each sprint lasted approx. 2 weeks, with some requirements changing at the end of each sprint, some of them shifting and some of them being removed completely (or added). The initial sprints consisted of mainly deciding the technical requirements for the plugin, the scope of the project etc. After that, began the phase of developing the plugin, where requirements were sometimes altered. After the major development, some changes were introduced to the plugin and the final exported documents were tested mostly by Adrian and me. The last phase was of course, documenting the project.

C.3 Workflow

The application was coded using the Apache NetBeans IDE, which is an integrated development environment for Java. It was very convenient for this plugin because of the debugging feature and adding dependencies and libraries was very straight-forward.

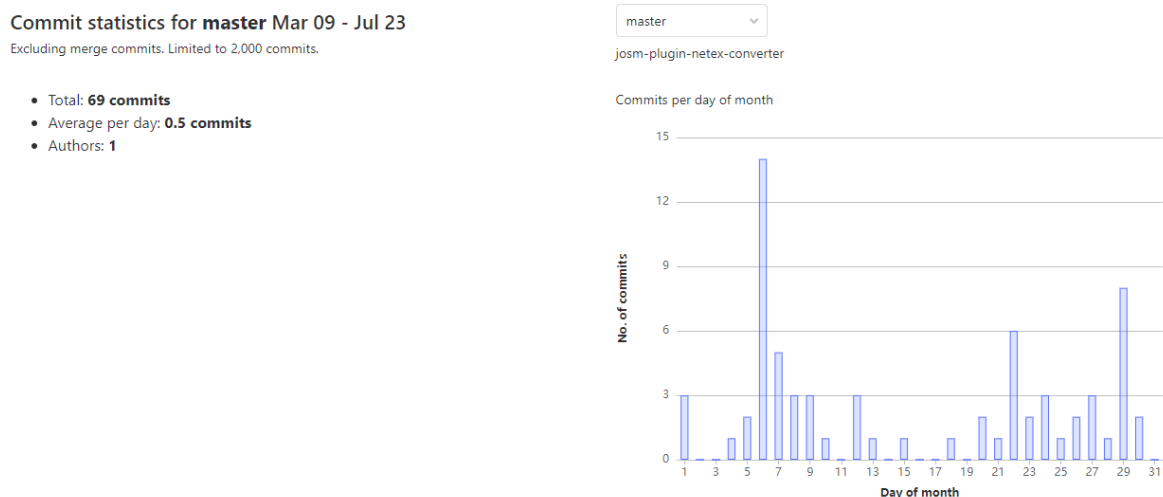The code repository and the CI/CD pipeline tools used for this plugin was GitLab.

FIGURE C.1: Some statistics about GitLab code commits (Commits per day are a little outdated)

CI / CD Analytics

Overall statistics

- Total: **62 pipelines**
- Successful: **46 pipelines**
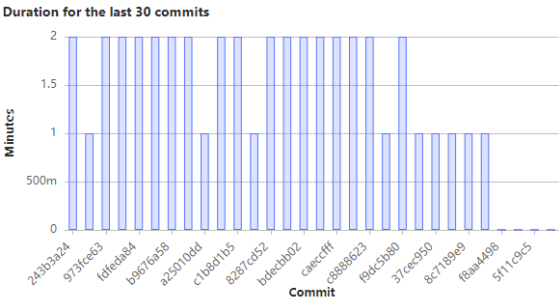- Failed: **16 pipelines**
- Success ratio: **74%**

FIGURE C.2: Some statistics about GitLab CI/CD pipelines

Bibliography

[1] Packt. What is openstreetmap? URL https://subscription.packtpub.com/book/hardware_and_creative/9781847197504/1/ch01lvl1sec06/what-is-openstreetmap. [Last Accessed: 18.07.2020].

[2] JOSM Community. Josm wiki, . URL https://josm.openstreetmap.de/. [Last Accessed: 20.07.2020].

[3] European Committee for Standardization CEN. Overview — netex. URL http://netex-cen.eu/?page_id=11. [Last Accessed: 20.07.2020].

[4] Parewa Labs Pvt. Ltd. Learn java programming, . URL https://www.programiz.com/java-programming. [Last Accessed: 21.07.2020].

[5] Parewa Labs Pvt. Ltd. Java program to find all roots of a quadratic equation, . URL https://www.programiz.com/java-programming/examples/quadratic-roots-equation.

[6] JOSM Community. Josm/plugins, . URL https://wiki.openstreetmap.org/wiki/JOSM/Plugins. [Last Accessed: 22.07.2020].

[7] OSM Community. Osm elements, . URL https://wiki.openstreetmap.org/wiki/Elements. [Last Accessed: 22.07.2020].

[8] Oracle Corporation. Lesson: Introduction to jaxb. URL https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html. [Last Accessed: 23.07.2020].

[9] Adam Kolawa Dorota Huizinga. *Automated Defect Prevention: Best Practices in Software Management.* Wiley-IEEE Computer Society Press, 2007. ISBN 0471491101.

[10] LaTeX3 Project. Latex – a document preparation system. URL https://www.latex-project.org/. [Last Accessed: 22.07.2020].

[11] Texmaker. Welcome to texstudio. URL http://texstudio.sourceforge.net/. [Last Accessed: 22.07.2020].

[12] Sascha Frank Uni Freiburg. Examples of latex. URL `http://www2.informatik.uni-freiburg.de/~frank/ENG/latex-course/latex-course-1/examples_en.html`. [Last Accessed: 23.07.2020].