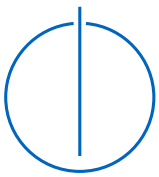
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

An Analysis of Traffic Simulations Based on Modifiable OpenStreetMap Data

Jakob Smretschnig



FAKULTÄT FÜR INFORMATIK

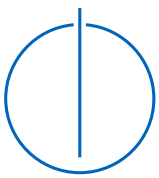
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

An Analysis of Traffic Simulations Based on Modifiable OpenStreetMap Data

Eine Analyse von Verkehrssimulationen mit modifizierbaren OpenStreetMap Daten

Author:	Jakob Smretschnig
Supervisor:	Prof. Bernd Brügge, Ph.D.
Advisor:	Mariana Avezum
Date:	16.09.2019



I assure the single handed composition of this bachelor thesis only supported by declared resources,

Munich, 16.09.2019 Jakob Smretschnig

f

Acknowledgements

I am using this opportunity to express my gratitude to my advisor Mariana Avezum, who supported me throughout the course of this thesis. Furthermore, I would like to thank my supervisor Prof. Bernd Brügge, Ph.D. for the trust he placed in me that made this thesis possible.

I am also grateful to Tobias Schock, head of the Economic Development of the municipality Kirchheim and his *IHK (German: Industrie- und Handelskammer) Smart Mobility* team who allowed me to work on a real problem. In addition, I would like to thank Michael Karrasch and Beatrix Winkler, both responsible for mobility and projects in Kirchheim, for their time and input throughout the evaluation.

All the assistance and valuable comments received during the course of this thesis greatly improved the implementation and the manuscript.

Finally, I take this opportunity to express my gratitude to all the members of the *Chair for Applied Software Engineering* at TU Munich for their help and support.

Abstract

Urban traffic poses a major challenge for governments and city planners. Traffic simulation software applications help to meet this challenge of enormous demand on mobility by providing ways to model and understand traffic flows and hence predict and improve transportation. However, the integration of OpenStreetMap (OSM) data into traffic simulation applications is limited, though OSM provides an accurate, up-to-date and open road network basis that covers most of our planet.

This thesis introduces an interactive system that seamlessly integrates the functionality of modifying OSM street layouts in a specific traffic simulation software by applying model based software engineering. To extract knowledge about highly congested areas, temporal traffic load maps are generated and visualized based on the simulation output. The overall goal of the system is to enable easy editing and quick simulation of traffic scenarios. Therefore transportation planning can be made as responsive, dynamic and agile as required by today's real world's necessities of an equally dynamic and constantly changing urban area.

The local authority of Kirchheim, a small municipality close to Munich, plans to use the developed system as part of its smart-city program to investigate novel transportation strategies. Kirchheim has been growing rapidly over the past years and due to a rising number of vehicles, complaints about noise, air pollution and traffic jams have become more frequent. Thus, a congestion-free traffic concept has the potential to improve Kirchheim's quality of life in a sustainable way.

Zusammenfassung

Städtischer Verkehr stellt eine der größten Herausforderungen für Politik und Wissenschaft dar. Verkehrssimulationen helfen dabei dieser Herausforderung entgegen zu treten indem sie Möglichkeiten bereitstellen, Verkehrsflüsse zu modellieren und zu verstehen und dadurch vorherzusagen und zu verbessern. Dennoch existierten kaum Verkehrssimulationsanwendungen, welche eine umfangreiche Arbeit mit OpenStreetMap (OSM) Daten unterstützen, obwohl OSM ein genaues, zeitgemäßes und frei verfügbares Straßennetzwerk der ganzen Welt bereitstellt.

Diese Arbeit präsentiert ein interaktives System, welches die Modifizierung von OSM Straßennetzwerken nahtlos mit einer ausgewählten Verkehrssimulationssoftware verknüpft. Dabei wird modellbasierte Softwareentwicklung angewendet. Um verkehrsreiche Gegenden und überlastete Straßen zu identifizieren, werden basierend auf dem Simulationsergebnis Karten mit dem Straßenverkehrsaufkommen generiert und visualisiert. Ziel des Systems ist es, einfaches Editieren und Simulieren von Verkehrsszenarien zu ermöglichen. Dadurch können Verkehrsplanungen möglichst dynamisch und agil durchgeführt werden, um den heutigen Notwendigkeiten eines ebenso dynamischen und sich ständig verändernden urbanen Raums zu entsprechen.

Die Gemeinde Kirchheim bei München plant das entwickelte System als Teil ihrer Smart-City Offensive einzusetzen um neue Transportstrategien zu erforschen. Kirchheim ist in den vergangenen Jahren stark gewachsen und durch die steigende Zahl an Autos häufen sich die Beschwerden über Lärmbelastung, Luftverschmutzung und Verkehrsstau. Ein staufreies Verkehrskonzept birgt daher das Potential, Kirchheims Lebensqualität nachhaltig zu verbessern.

Contents

1 Introduction **1**
- 1.1 Problem Statement . . . 1
- 1.2 Background . . . 2
- 1.3 Objectives . . . 2
  - 1.3.1 Realistic Traffic Demand . . . 2
  - 1.3.2 Editor Integration . . . 3
  - 1.3.3 Traffic Flow Visualization . . . 3
- 1.4 Motivation . . . 3
- 1.5 Overview . . . 4
- 1.6 Outline . . . 4

2 Background **5**
- 2.1 Pilot Projects vs. Computer Simulations . . . 5
  - 2.1.1 Computer Simulations . . . 5
- 2.2 Traffic Simulation Software . . . 6
  - 2.2.1 Simulation Strategies . . . 6
  - 2.2.2 Multimodal Transport Simulation . . . 6
  - 2.2.3 Simulation Input . . . 7
  - 2.2.4 SUMO - Simulation of Urban MObility . . . 8
  - 2.2.5 CityMos - City Mobility Simulator . . . 9
  - 2.2.6 MATSim - Multi-Agent Transport Simulation . . . 10
- 2.3 OpenStreetMap . . . 10
  - 2.3.1 Attributes . . . 10
  - 2.3.2 JOSM - Java OpenStreetMap Editor . . . 12
- 2.4 Modeling Mobility with Open Data . . . 12
- 2.5 Coordinate Reference Systems . . . 13
  - 2.5.1 GPS Reference System . . . 13
  - 2.5.2 Map Projection . . . 14
- 2.6 Scrum . . . 15

3 Requirements Elicitation **17**
3.1 Overview . . . 17
3.2 Functional Requirements . . . 17
3.3 Nonfunctional Requirements . . . 18
3.4 Use Case Diagram . . . 19
3.5 Scenarios . . . 20

4 System Design **25**
4.1 Analysis Object Model . . . 25
4.1.1 Stereotypes . . . 27
4.2 Subsystem Decomposition . . . 27
4.3 Dynamic Behavior . . . 28
4.4 Graphical User Interface . . . 30
4.4.1 Simulation Model . . . 30
4.5 Design Goals . . . 31
4.5.1 Chosen Traffic Simulation . . . 31
4.5.2 Chosen OpenStreetMap Editor . . . 31
4.5.3 Trade-Offs . . . 33
4.6 Identify Concurrency . . . 34
4.7 Hardware/Software Mapping . . . 35
4.7.1 Container Platform . . . 35
4.8 Persistent Data Management . . . 36

5 Agile City Planning Suite **37**
5.1 Overview . . . 37
5.2 Object Design . . . 38
5.2.1 Stereotypes . . . 40
5.2.2 Object Description . . . 40
5.3 JOSM Container . . . 42
5.4 TraLAMA Plugin . . . 42
5.4.1 Filter OSM Elements . . . 43
5.4.2 Communication with SUMO Container . . . 44
5.4.3 Generation of Traffic Load Maps . . . 45
5.4.4 TraLAMA Workflow . . . 45
5.5 SUMO Container . . . 47
5.5.1 Simulation Controller . . . 47
5.5.2 Route Planner . . . 50
5.6 TraLAMA Web Service . . . 53
5.7 User Workflow . . . 54

6 Evaluation **57**
6.1 Objectives . . . 57
6.2 Methodology . . . 57
6.3 Simulation Scenarios . . . 58
6.4 Results . . . 58
6.4.1 Scenario 1 - Original Road Network . . . 58
6.4.2 Scenario 2 - Bypass Road . . . 59
6.4.3 Scenario 3 - Bicycle Road . . . 60
6.4.4 Usability . . . 60
6.4.5 Traffic Load Map . . . 61
6.4.6 Sensor Data . . . 61
6.4.7 Multimodal Simulations . . . 61
6.4.8 Summary . . . 61
6.5 Findings and Discussion . . . 62
6.5.1 Street Classification . . . 62
6.5.2 Usability . . . 62
6.5.3 Web Service, Sensors and Bicycles . . . 62
6.6 Reliability . . . 63

7 Summary **65**
7.1 Status . . . 65
7.1.1 Realized Goals . . . 65
7.1.2 Open Goals . . . 68
7.2 Threats to Validity . . . 68
7.3 Future Work . . . 69
7.4 Conclusion . . . 70

A Software Availability **71**
A.1 Prerequisites . . . 71
A.1.1 Windows . . . 71
A.1.2 macOS . . . 72
A.2 Installing . . . 72
A.3 Starting . . . 72
A.3.1 Windows . . . 72
A.3.2 macOS . . . 72
A.4 Additional Information . . . 73
A.4.1 Ports . . . 73
A.4.2 Data . . . 73
A.5 References . . . 73

Bibliography **82**

ACPS Agile City Planning Suite

AOM Analysis Object Model

API Application Programming Interface

CSV Comma-Separated Values

DLR German Aerospace Center (German: *Deutsches Zentrum für Luft- und Raumfahrt e. V.*)

EPL Eclipse Public License

GIS Geographic Information System

GNSS Global Navigation Satellite Systems

GPL GNU General Public License

GPS Global Positioning System

GPX GPS Exchange Format

GRS Geodetic Reference System

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

IHK Chamber of Industry and Commerce (German: *Industrie- und Handelskammer*)

IP Internet Protocol

ITRS International Terrestrial Reference System

JOSM Java OpenStreetMap Editor

JSON JavaScript Object Notation

ODbL Open Data Commons Open Database License

OSI Open Systems Interconnection

OSM OpenStreetMap

POI Point Of Interest

REST REpresentational State Transfer

SUMO Simulation of Urban MObility

TAZ Traffic Assignment Zones

TraCI Traffic Control Interface

TraLAMA Traffic LoAd MAp

TUM Technical University of Munich (German: *Technische Universität München*)

UML Unified Modeling Language

URL Uniform Resource Locator

UTM Universal Transverse Mercator

WGS World Geodetic System

XML Extensible Markup Language

Chapter 1

Introduction

The number of people moving from rural to urban areas is rising [The18, Uni18]. The reasons for this range from a lucrative labor market to better infrastructure and improved health care [JCG14]. This shift results in enormous changes in city dynamics and governments and city planners are challenged in many ways. One major challenge is transportation, since people need to fulfill their essential human need of mobility. Historically, the transport infrastructure was not designed to handle such a large number of people. For instance, the central suburban line (S-Bahn) in Munich, which was originally built in 1972 for about 250,000 passengers per day, nowadays carries up to 840,000 passengers on a business day [Deu17]. The amount of registered cars in Germany has also increased tremendously: from 1990 to 2019 by more than fifty percent to a total of about 47.1 million today [Sta19]. In other words, more than every second citizen owns a car. As a result, many citizens suffer from air pollution, noise pollution and a high level of stress caused by traffic [Wor18].

1.1 Problem Statement

Kirchheim, a small municipality close to Munich, has been growing rapidly over the past years too and Kirchheim's streets are typically congested during rush hour. If the municipality ignores the problem, not only the livability in Kirchheim gets worse, but also the industrial sector might suffer from a lack of specialists, who would rather choose a workplace in other regions.

Based on this problem, the goal is to support the municipality of Kirchheim in developing a congestion-free traffic concept. Therefore, open data and freely available traffic simulations are used to analyze the impact of road network editing on traffic flow. Besides, commuter streams and traffic statis-

tics of Kirchheim are considered to model the simulation scenarios as realistic as possible.

1.2 Background

In fact, the first traffic simulations date back to the 1950s [Tra14]. Thus, several traffic simulations already exist that allow city and traffic planners to develop and assess novel transportation strategies. To simulate routes of vehicles on a given road network, a traffic simulation usually requires two types of input data: a road network, representing the traffic areas and junctions and traffic demand, describing the movements of vehicles [Bar10, BW15]. Both datasets have to be realistic to obtain a meaningful simulation result.

Road networks could be provided by government agencies. However, the maintenance is difficult and the networks might be delivered in different file formats, depending on the authority [ZNN11]. Since millions of hand-held navigation devices collect high-quality location information nowadays and thousands of users worldwide who own such a device share their location information, the collaborative OpenStreetMap (OSM) project was founded [HW08]. OSM provides an accurate, up-to-date and open road network basis that covers most of our planet. Soon it emerged as a state-of-the-art data source to derive suitable road networks for traffic simulations.

1.3 Objectives

1.3.1 Realistic Traffic Demand

Obtaining realistic traffic demand data is complex [BW14]. Therefore, traffic demand is often generated randomly within a simulation. However, in order to analyze the impact of road network editing on traffic flow, the traffic demand must not be changed. This means that the origin and destination locations of all vehicles have to be constant throughout different simulation scenarios with modified road networks. Within this thesis, tools are implemented to store the traffic demand independently of the traffic simulation and to reuse the demand data for the calculation of new simulation routes, based on a modified road network.

1.3.2 Editor Integration

In order to edit the road network, for example by adding more lanes, a new street, traffic lights or traffic signs to it, the traffic simulation SUMO for instance integrates a tool called NETEDIT[1]. However, this tool is not as powerful as external editors such as JOSM (Java OpenStreetMap Editor)[2] or GIS (Geographic Information System) applications like QuantumGIS[3] or ArcGIS[4] are. To derive benefit of JOSM's range of functions, its user-friendly interface and large community, another goal is to integrate the editor and a specific traffic simulation in order to combine their functionalities. In addition, by fully automating all required processing steps that are required for the integration, the entire system can be utilized with basic computer knowledge.

1.3.3 Traffic Flow Visualization

In order to analyze the impact of road network editing on traffic flow, the traces of the simulated vehicles should be visualized in the form of a heat map. These traffic load maps should enable traffic planners to extract knowledge about highly congested areas in a simple way.

1.4 Motivation

The primary goal of the system is to provide traffic planners with a simple way of experimenting visionary transportation scenarios based on modified road networks. Thereby, benefit is derived from accurate and up-to-date OSM data. Moreover, traffic planners get instant feedback through the temporal traffic load maps and can understand, analyze and optimize the traffic flow more easily. As a result, novel transportation strategies can be assessed more efficiently.

Since the system is developed on behalf of the municipality Kirchheim, it is a small step towards a congestion-free traffic concept that has the potential to tremendously improve Kirchheim's quality of life.

[1] `https://sumo.dlr.de/wiki/NETEDIT`
[2] `https://josm.openstreetmap.de`
[3] `https://www.qgis.org/en/site/`
[4] `https://www.esri.com/en-us/arcgis/about-arcgis/overview`

1.5 Overview

Within this Bachelor's Thesis, the following systems were implemented:

1. **Agile City Planning Suite** (ACPS)
2. **TraLAMA** (Traffic LoAd MAp) **Plugin**
3. **TraLAMA Web Service**

The *Agile City Planning Suite* describes the entire system which includes the TraLAMA plugin, the TraLAMA web service and an additional OpenStreetMap (OSM) editor as well as a traffic simulation software.

The TraLAMA plugin is a plugin for the OSM editor and allows traffic planners to use modified OSM data as a road network basis for the traffic simulation. The plugin further includes optimized traffic demand data of the municipality Kirchheim to enable more precise simulations of this area than with randomly generated data. The simulation results can be visualized in the OSM editor as additional layers and on the web page provided by the TraLAMA web service, which offers extended styling options.

1.6 Outline

In Chapter 2, the background technologies of traffic simulations and OpenStreetMap will be discussed. Then the system requirements will be derived in Chapter 3, based on the problem statement. These requirements will be later used for the system design in Chapter 4, where they are mapped to design goals and subsystems. Chapter 5 focuses on the solution domain and the actual implementation. The subsequent Chapter 6 presents several simulation scenarios that were tested among municipal employees of Kirchheim. Lastly, Chapter 7 summarizes open and realized goals of the implemented systems and draws a conclusion from this thesis.

Chapter 2

Background

This chapter provides insight into computer simulations in general (2.1), state-of-the-art traffic simulation software (2.2) and OpenStreetMap (OSM) editors (2.3). Existing works about *Modeling Mobility with Open Data* (2.4) are discussed and a brief introduction into the agile software development methodology *Scrum* is provided (2.6).

2.1 Pilot Projects vs. Computer Simulations

Usually real world pilot projects are expensive to realize and their impact is difficult to reproduce. That is because pilot projects need large effort for planning and run for several days, weeks or even months and years. Obviously, the longer the testing phase, the more accurate are the results. On the contrary, long testing phases represent the exact opposite of agile development, where changes happen frequently and quick response has a high priority [BD09]. Thus, one could argue that pilot projects are always behind the real world's necessities. To measure the impact of a pilot project, one also needs previous data for the testing area to determine changes, whether the experiment effects the area with a desirable or an undesirable impact. Consequently, the required time for the testing phase could increase even more [KH09].

2.1.1 Computer Simulations

Constant progress in computer architecture and computer processing power enables the manufacturing of unprecedented high-performance computers, so-called supercomputers. These supercomputers let scientists solve complex computations within reasonable time. One field of application is the

computer simulation, a discipline of high relevance in science. With the aid of intricate computer simulations, experts can test scientific models for its feasibility and correctness faster and at lower cost than conducting the experiment in real world [KH09].

2.2 Traffic Simulation Software

Throughout the last centuries, research in the field of mobility and traffic management got more attention where the simulation of traffic networks to calculate an optimal workload of the road users has played an important role [KH09].

2.2.1 Simulation Strategies

Scientists distinguish between three traffic simulation strategies [Bar10,BW15, KH09]:

- Macroscopic Traffic Simulation: Vehicles are statistically distributed over the road network
- Microscopic Traffic Simulation: Any road user (e.g. cars, motorbikes, bicycles), pedestrians and public transport can be modeled explicitly.
- Mesoscopic Traffic Simulation: Represents an intermediate of micro- and macroscopic simulation strategies. According to Barceló [Bar10], it is "the simplification that intends to capture the essential points of the dynamic, while demanding less data and hence is computationally more efficient than microscopic models".

Especially the microscopic traffic simulations take advantage of the high-performance computers. More precise simulations can be modeled that reflect the dynamic city behavior in a more realistic approach than ever before. Figure 2.1 illustrates the different traffic simulation strategies.

2.2.2 Multimodal Transport Simulation

Traffic simulations that involve different modes of transport are defined as multimodal transportation simulations [GD12]. This means that the simulations are not only based on vehicles, but on a combination of vehicles, bicycles and pedestrians for instance.

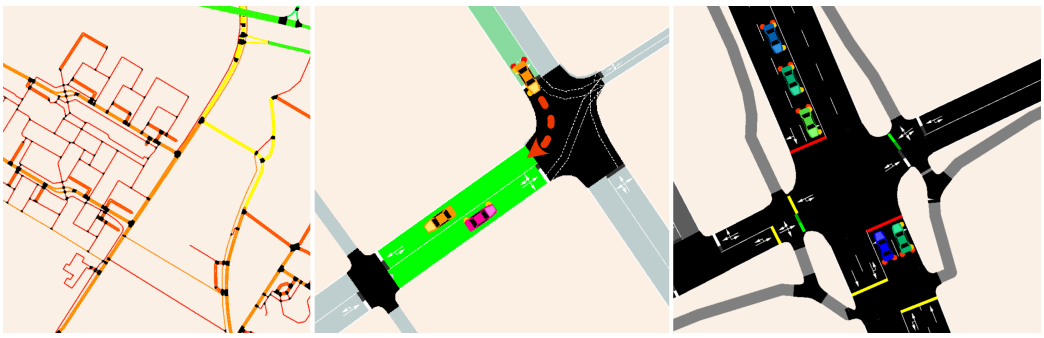


Figure 2.1: Traffic Simulation Strategies: Macroscopic, Mesoscopic, Microscopic

Figure 2.2: Road Network of Kirchheim

2.2.3 Simulation Input

To simulate routes of vehicles on a given road network, a traffic simulation usually requires two types of input data: the road network and traffic demand data [Bar10,BW15]. Both datasets have to be realistic to obtain a meaningful simulation result.

Road Network

The road network represents the road structure within the traffic simulation. This means it includes the traffic areas and junctions, with additional information about the number of lanes, speed limits, street types and so on. Figure 2.2 shows an example of a road network.

Traffic Demand

Unlike the road network, the traffic demand data usually differs from one simulation strategy (2.2.1) to another. Given a macroscopic traffic simulation, the traffic demand describes the traffic flows within the simulation. These flows are often provided by traffic authorities and represented as *Orign-Destination-Matrices* [BBEK11]. Whereas on a microscopic level, individual movements of road users such as vehicles, bicycles or pedestrians are modeled. Using a *trip* definition, the origin and destination locations are stored for a specific road user object. An example would be a simple table structure, with the following columns: *Vehicle ID*, *Start Time*, *Origin Geo-Coordinate*, *Destination Geo-Coordinate*.

Typical data sources for the traffic demand are traffic sensors and statistics about the mobility behavior of people [Bar10]. The statistical data is often estimated based on information about commuter flows, population density and company locations together with their number of employees.

2.2.4 SUMO - Simulation of Urban MObility

SUMO is a free and open-source traffic simulation suite that allows traffic planners to run a microscopic or mesoscopic traffic simulation [BBEK11]. The road network itself is a graph that can be either modeled manually or generated out of existing models, e.g. OSM models using the NETCONVERT tool. Besides, SUMO allows the user to perform multimodal traffic simulations (2.2.2) with vehicles, pedestrians, public transport and many more [KHRW02, BBEK11, Bar10]. It is licensed under the *Eclipse Public License v2.0* (EPLv2)[1].

NETCONVERT

NETCONVERT is a conversion tool that provides traffic planners with an option to import road networks from different sources [KHRW02]. It is a command-line application and hence, requires a few basic computer skills. NETCONVERT takes an OSM file as input data and stores the SUMO road network in a new file. To customize the conversion process, additional parameters can be added such as *Output Street Names* or *Junctions Join*. With the former option, the generated SUMO road network includes street names. The latter option is especially recommended for OSM imports. Often, traffic lights are modeled several times for one junction because they only represent their position within the OpenStreetMap platform. SUMO on the other

[1] `https://www.eclipse.org/legal/epl-2.0/`

hand precisely requires one traffic-light element per junction. Therefore, the *Junctions Join* option takes care that the traffic lights get synchronized for an appropriate mapping to the SUMO road network [KHRW02].

NETEDIT

NETEDIT is part of the SUMO package and can be used to create new SUMO road networks from scratch or modify existing ones. The tool is built on top of NETCONVERT and features almost the same functionalities. Unlike NETCONVERT, NETEDIT offers a graphical user interface [KHRW02].

POLYCONVERT

POLYCONVERT is a SUMO tool that parses OSM points of interest (POI) and additional geometric shapes such as farmlands or buildings for the visualization in SUMO. In order to realize that, a configuration file is needed in the form of an XML (Extensible Markup Language) document. This document represents a schema definition for the parsing of OSM elements into SUMO elements. It can be customized to let the user decide on how much additional information is shown in the simulation. POLYCONVERT writes the result into a SUMO *shape file*, which can be then read by SUMO as an additional simulation input file [KHRW02].

Trips and Routes

A SUMO *trip* describes the starting time, the starting edge and the destination edge of a vehicle. Notable is that the vehicles do not start from nodes which would represent precise locations, but are rather placed randomly along the edges. Random trips can be generated using SUMO's *Random Trips* script. To simulate the vehicle along the streets of the road network, the trip information has to be complemented by the actual *route* definition. *DUAROUTER*, another command-line application of the SUMO package, computes the intermediate edges a vehicle will pass when driving from the given starting edge to its destination edge. For the route calculation, Dijkstra's shortest path algorithm is used per default [KHRW02].

2.2.5 CityMos - City Mobility Simulator

The *City Mobility Simulator* (CityMos) is another traffic simulation with a focus on modeling both conventional and autonomous vehicles [ZNKE17]. CityMos is a (sub-)microscopic simulation that considers the components of the vehicles and even the behavior of the driver. It is not possible to

import OSM data directly, however there exists a *scenario editor* with the functionality to create CityMos networks based on OSM. CityMos is still under development and not available to the public yet.

2.2.6 MATSim - Multi-Agent Transport Simulation

The MATSim project is an open-source simulation framework to model large-scale scenarios. Typically, a single day is modeled and simulated. Thereby, MATSim simulates microscopic agents [HNA16].

2.3 OpenStreetMap

OpenStreetMap (OSM) is a collaborative project that was launched in 2004 where thousands to millions of users share their location information to build a free and editable worldwide map [HW08]. The founder Steve Coast got the inspiration to such a volunteered map from the free encyclopedia Wikipedia. Unlike Wikipedia, where everyone can be an honorary and anonymous author, only registered users are allowed to edit the OSM maps. Besides the main web page[2] and its online editor, there are several editing tools such as the JOSM editor, which is described in Section 2.3.2.

OpenStreetMap is licensed under the *Open Data Commons Open Database License* (ODbL)[3] which means that anybody can copy and adapt OSM data as long as they give credit to OpenStreetMap and its contributors [Ope19a].

2.3.1 Attributes

There exists no directive how to tag OSM elements correctly, the consensus evolved by the users themselves [HW08]. Of course there are some guidelines and best practices, but still anybody can freely add a new *key=value* pair to any OSM element. This fact makes it difficult to define a basis between the OSM data and the traffic simulation software.

Street Types

The main attribute of streets in OSM has the key *highway*. The keywords listed in Table 2.1 are relevant for the correct classification of streets [Ope19d]. They are ordered by priority.

[2] `https://www.openstreetmap.org`

[3] `https://opendatacommons.org/licenses/odbl/`

OSM	German	Description
motorway	*Autobahn*	divided highway with restricted access
trunk	*Kraftfahrstraße*	similar to the motorway, but not always structurally separated
primary	*Bundesstraße*	main transport connection between large towns
secondary	*Landesstraße*	nationwide importance, links towns
tertiary	*Kreisstraße*	links smaller towns or villages
unclassified	*Gemeindestraße*	usually no center line
residential	*Wohnstraße*	provides access to housing
living street	*Verkehrsberuhigte Straße*	often called *play street*, pedestrians have priority over cars

Table 2.1: Street Type Classification in OpenStreetMap

Attention: The road tag *unclassified* is not a placeholder. It is a term that has a historical background in the United Kingdom. If the classification is unknown, it should be tagged with *road*.

Junctions

The following values should be used for the OSM attribute *junction* [Ope19e]:

- **roundabout**: a circular road junction which always has right of way
- **circular**: a road junction which does not always have right of way
- **jughandle**: a road junction including ramps

All three junction types require the additional attribute $highway = *$, with $*$ representing one of the street types (2.3.1). Otherwise, it is probably a standalone junction that does not connect highways yet. Such a junction should be tagged as $junction = yes$.

An additional tag for a junction could be $supervised = yes$, representing a police officer who regulates the traffic.

2.3.2 JOSM - Java OpenStreetMap Editor

JOSM is an open-source offline editor for OpenStreetMap that supports instant downloading of OSM material from the OpenStreetMap server and aerial imagery to get a better overview of the scene. JOSM is also capable of working with GPX (GPS Exchange Format) files and provides a multilayer functionality. The editor has various built in presets for OSM elements such as street types, traffic signs and almost any facility one can imagine [HW08]. JOSM is released under the *GNU General Public License* (GPL)[4].

2.4 Modeling Mobility with Open Data

During the second SUMO Conference in Berlin in 2014, participants were dealing with one specific topic: *Modeling Mobility with Open Data* [BW14]. The focus of the proceedings of the conference was on free data sources for traffic simulations. Several applications and projects were presented that are based on the traffic simulation SUMO.

For instance, the authors of [RSR15] introduced a plugin for JOSM to enable advanced editing of traffic lights. The proposed *Traffic Signals Editor* should make it easier to tag complex intersections appropriately for the reuse in SUMO. Thereby, the OSM format is extended with a *traffic signal* relation and additional attributes are stored as key-value pairs. Unfortunately, it seems that the developed plugin was not published yet, since it is not mentioned in the list of all plugins for JOSM[5].

Another study by [KP15] presents calibration steps to improve traffic demand gathered through sensor data. In the Austrian states of Salzburg and Upper Austria, several traffic sensors were installed to measure traffic flow. However, these sensors do not cover the entire road network. In order to fill the gaps of traffic demand, the authors use the traffic simulation SUMO to generate additional demand data. Thereby, the simulation results are compared with the real-time data to adjust the simulation settings automatically.

The contributors of [KBW+15] address air pollution emitted by vehicles. They implemented two new models for SUMO in order to compute CO, CO_2, NO_x and other emissions of simulated vehicles. To calculate these emissions for each road user individually, a microscopic traffic model was used.

[4] `https://www.gnu.org/licenses/gpl-3.0.en.html`
[5] `https://josm.openstreetmap.de/wiki/Plugins`

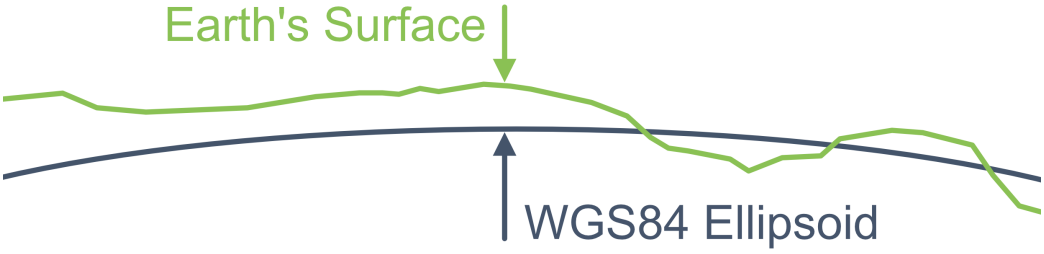


Figure 2.3: *WGS84* Ellipsoid Showing the Difference in Elevation

2.5 Coordinate Reference Systems

Coordinate reference systems are used to "determine events in space and time" [Lan19b]. Therefore, the coordinate reference system is defined through coordinates with regard to a specific spatial reference system. It is often also referred to as *direct spatial reference*. Whereas the *indirect* spatial reference uses geocoding: a simple addressing scheme such as *Arcisstraße 21, 80333 Munich*, which can be then decoded to obtain a direct spatial reference. In the following, the focus is put on the direct spatial reference only.

To describe the coordinate reference system, three components are necessary:

- **Geodetic Reference System**: Is defined through the *geodetic datum* that approximates the earth with a global or national adapted ellipsoid and the associated mounting point of this ellipsoid. In Figure 2.3, the approximation using an ellipsoid is illustrated.
- **Reference Frame**: Precise definition of fixed points in the earth's terrain to realize the reference between the geodetic reference system and the real world.
- **Coordinate System**: Different types such as the geocentric Cartesian coordinates (typically used for satellite orbits), geographic coordinates (latitude, longitude) or projection coordinates (described in Section 2.5.2) exist.

2.5.1 GPS Reference System

The **Global Positioning System** (GPS) is the most known *Global Navigation Satellite System* (GNSS) [Lan19a]. It is properly specified with the geodetic reference system *WGS84* (World Geodetic System 1984) and the

reference frame *WGS84*. GPS has almost the same characteristics as the *International Terrestrial Reference System* (ITRS), which uses the *GRS80* (Geodetic Reference System 1980) ellipsoid and is mounted in the geocentric center of mass of the earth. Besides, GPS data is usually provided in the form of geographic coordinates. If needed, they can be translated to three-dimensional Cartesian coordinates, using mathematical operations.

2.5.2 Map Projection

A map projection is used to map three-dimensional points from a curved surface such as the earth to a two-dimensional plane surface, for instance a display or a printed map. Therefore, two mapping approaches exist: *Peeling*, whereby the plane surface is not covered completely and blank areas occur, and *reverse mapping*, where for each point of the plane a point of the three-dimensional object is being assigned. The latter approach is the most commonly used one, though it leads to distortions [Lan19b].

One example would be the *Mercator* projection, which is conformal (angle preserving). However, the size of objects away from the equator is skewed. Probably anybody who has seen a Mercator world map once has therefore raised the question whether Greenland has indeed the size of Africa, which is not the truth.

Another projection type is the equal-area projection by *Gall-Peters*. The disadvantage with this projection is that countries and continents away from the equator are illustrated with wrong angles, giving them an unrealistic and unusual look. Imagine a navigation system, where a straight highway appears as a curved line. In brief, each projection type has its limitations when mapping the surface of a whole sphere onto a plane surface.

Yet, sometimes it is required to only map a certain area of the earth, such as a state or a country. Therefore, the *Universal Transverse Mercator* (UTM) coordinate system was developed. As the *Mercator* projection, UTM is a conformal map projection and is based on the global *GRS80* ellipsoid. UTM uses a transversal cylinder to partition the globe into sixty zones of 6 degrees of longitude. Coordinates are indicated using an easting value and a northing value. In Germany, the authoritative real estate cadastre (German: *Amtliches Liegenschaftskataster*) uses the UTM map projection together with the *European Terrestrial Reference System* (ETRS89), the European part of the ITRS.

2.6 Scrum

Scrum is a synonym for agile software development, where the focus is put on the individuals, a working software, customer collaboration and responding to change [BD09]. This is relevant for software development because both requirements and technology may change rapidly. Therefore, the Scrum methodology is based on little incremental steps and iteration. Another characteristic of Scrum is the fact that not all pieces of work are completely understood at the beginning.

All these characteristics apply to this work and based on that, the Scrum methodology was chosen for the development process of this thesis. Thereby, the three major components *Activities*, *Artifacts* and *Roles* [BD09] were partially incorporated:

- **Scrum Activities**
  Throughout the *Sprints* (iterations), informal weekly (instead of daily) Scrum meetings helped to get an overview of the current status of the project. Discussed issues were previous work, impediments and promises. After each *Sprint*, the implemented features were presented to the customer within the *Sprint Review Meeting*. These meetings took place in the city hall of Kirchheim.

- **Scrum Artifacts**
  The two main artifacts of Scrum are the *Product Backlog* and the *Spring Backlog*. The former describes open issues for the whole project, whereas the latter describes a subset of it for the current *Sprint*. Throughout this thesis, both backlogs were used.

- **Scrum Roles**
  The *Scrum Master* is responsible for the management of the project. In this case, the advisor. The customer who knows what has to be built is the *Product Owner*, which is the municipality Kirchheim. Lastly, there was only one developer, forming the *Scrum Team*.

The authors of [AEE15a] have also shown that Scrum is a proper methodology for the development of traffic simulation based applications. They used the Scrum methodology and considered the re-engineering process to develop tools for the traffic simulation SUMO.

Chapter 3

Requirements Elicitation

Initially, functional (3.2) and nonfunctional (3.3) requirements are derived from the problem statement (1.1). The subsequent use case diagram (3.4) illustrates the functional behavior of the system in the UML (Unified Modeling Language) notation. The diagram reduces complexity and hence provides a better communication point among stakeholders involved in this project [BD09]. The chapter concludes by describing two as-is scenarios and a visionary scenario (3.5) to summarize the behavior of the system as seen by a traffic planner.

3.1 Overview

The aim is to develop a system that seamlessly integrates an OpenStreetMap editor with a traffic simulation platform. The integration should be implemented in such a way that the system is usable by any authority and does not require advanced computer knowledge. This means that the workflow has to be intuitive and the simulation results must be represented comprehensibly.

3.2 Functional Requirements

FR1 **Load Road Network Data**: The system must provide a way to download and import OpenStreetMap data from the OpenStreetMap server. This enables a solid starting point for the data processing.

FR2 **Modify Road Network Data**: There must be an option to modify OpenStreetMap data, e.g. add new lanes, add traffic lights and other

transportation elements. Hence, various novel traffic strategies can be modeled.

FR3 **Export Road Network Data**: There must be a way to convert and export the modified OpenStreetMap data to the traffic simulation platform so that the novel traffic strategy can be tested.

FR4 **Select Traffic Demand**: Traffic planners must have the option to select from temporal different traffic demand data. Thereby, diverse traffic simulations can be generated using various time periods such as the morning or evening rush hour.

FR5 **Customize Traffic Simulation Settings**: The system should implement graphical user interface elements to allow traffic planners to modify the simulation length and other simulation settings.

FR6 **Run Traffic Simulation**: There must be a way to run a traffic simulation to evaluate the novel traffic strategy.

FR7 **Visualize Traffic Load Maps**: Temporal traffic load maps must be generated and visualized using the traffic simulation output. These maps make it easier for traffic planners to extract knowledge about highly congested areas and calm ones.

FR8 **Export Traffic Load Maps**: There must be an option to export the generated traffic load maps as images or in a vector format to share with stakeholders.

FR9 **Show Simulation Statistics**: Detailed simulation statistics must be created and visualized based on the traffic simulation output to provide insight into the average waiting time of vehicles and other measurable results.

3.3 Nonfunctional Requirements

NFR1 **Accuracy**: The road network should be up-to-date, contain the correct amount of traffic lanes and its directions as well as proper junctions, e.g. roundabout, traffic light settings. This is important, since the road network represents the basis of the traffic simulation.

NFR2 **Usability**: The entry point to the system should be enabled with a maximum of two clicks to ensure an easy way of starting the application.

NFR3 **Usability**: The graphical user interface and the traffic load map should be colorblind safe to include more potential system users.

NFR4 **Usability**: The interaction between the OpenStreetMap editor and the traffic simulation should be enabled with one click to ensure an easy handling.

NFR5 **Portability**: The system should be platform independent and able to run on Microsoft Windows and macOS in order that different workstations are able to execute the system.

NFR6 **Supportability**: The communication point between the OpenStreetMap editor and the traffic simulation should be based on the REST paradigm. This enables scalability and independence between the client and the server.

NFR7 **Response Time**: The creation process of the visualization of the simulation output must not take more than five minutes. Otherwise, the workflow would be too limited.

NFR8 **Packaging (Constraint)**: The system should be packaged in a lightweight container. Thereby it can be easily distributed on different workstations without the time-consuming setting up of environments that might differ from one computer to another.

NFR9 **Legal (Constraint)**: The system must be free and comply to the open-source standard to resolve potential anxiety of the corporatisation of the community authorities.

NFR10 **Interface (Constraint)**: The system must support the usage of OpenStreetMap data.

3.4 Use Case Diagram

As shown in Figure 3.1, the traffic planner has the following use cases:

- **Import Road Network**: The traffic planner is able to import an existing road network to have a basis to work with.
- **Modify Road Network**: The traffic planner can add bypass roads, new lanes, traffic lights, roundabouts and other transportation elements.

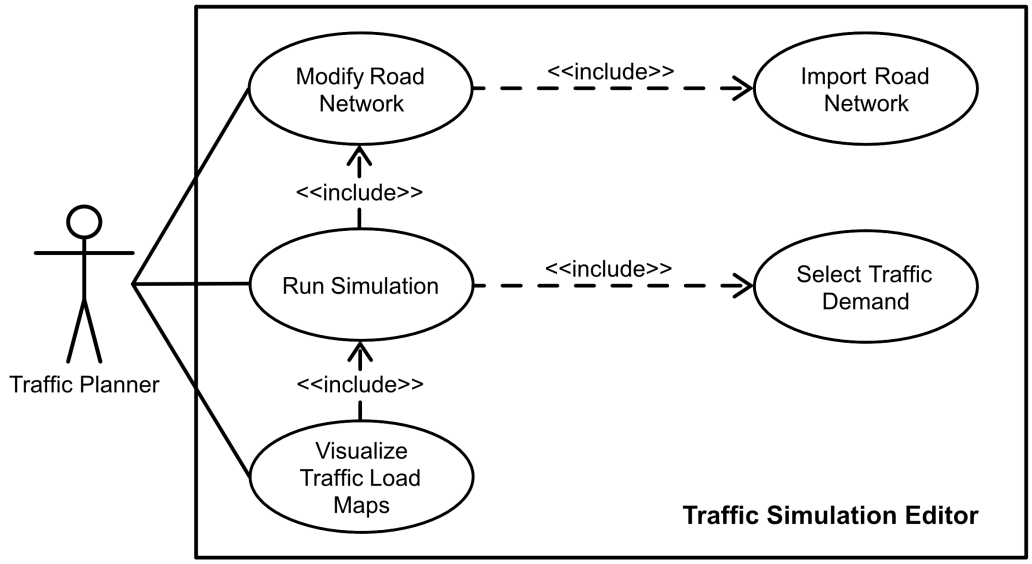


Figure 3.1: UML Use Case Diagram

- **Select Traffic Demand**: Since there exists different traffic volume during the morning, noon or evening rush hour, the traffic planner can select from different traffic demand data.
- **Run Simulation**: Using the (modified) road network data and the selected traffic demand information, the traffic planner can start a new traffic simulation.
- **Visualize Traffic Load Map**: Based on the simulation output, the traffic planner can see and review temporal heat maps, showing the traffic load. However, a traffic load map can only be visualized after the simulation.

Entry conditions for the use case diagram in Figure 3.1 are a running road network editor as well as a running traffic simulation platform. Both have to be connected to each other.

3.5 Scenarios

The following scenarios provide a glimpse into the state-of-the-art software and visionary scenarios. Table 3.1 describes how a traffic planner can prepare a novel traffic strategy by modifying the layout of the road network. Table 3.2 explains the workflow of the traffic simulation and Tabl 3.3 presents the visual output of the traffic simulation in the form of a heat map.

Scenario Name	Modify the Road Network
Participating Actors	Traffic planner Nathalie
Entry Conditions	1. Running map editor 2. Loaded OpenStreetMap data of *Kirchheim*
Flow of Events	1. Nathalie selects the junction in front of the *Rathaus* which includes *Münchner Straße* and *Heimstettner Straße*. 2. She places a traffic light there. 3. She further adds a bypass road to *Heimstettner Straße* because in previous simulations she found out that this street is highly congested most of the time. 4. Lastly, she performs a data validation to prove that the novel road network is modeled appropriately.
Exit Conditions	Successful data validation
Special Requirements	*None*

Table 3.1: As-is Scenario 1: Modify the Road Network

Scenario Name	Run a Traffic Simulation
Participating Actors	Traffic planner Nathalie
Entry Conditions	1. Running traffic simulation software 2. Loaded road network and traffic demand
Flow of Events	1. Nathalie presses the *Play* button to start the traffic simulation. 2. She increases the *delay* up to 170ms to slow down the simulation speed and be able to better analyze the traffic flow. 3. Nathalie also colors the vehicles based on their *waiting time* and street lanes based on their *current occupancy*. 4. Shortly after the last vehicle has reached its destination, the simulation stops. 5. Nathalie reviews the simulation statistics and finds out that none of the vehicles was involved in a collision.
Exit Conditions	All simulated vehicles have reached their destination.
Special Requirements	The road network file and the traffic demand file must be given.

Table 3.2: As-is Scenario 2: Run a Traffic Simulation

Scenario Name	Visualize Traffic Simulation Output
Participating Actors	Traffic planner Nathalie
Entry Conditions	1. The map editor is running 2. Running traffic simulation software
Flow of Events	1. Nathalie starts the traffic simulation by clicking the *Play* button. 2. When the simulation is done, a new layer is loaded into the map editor, showing a traffic load map of all vehicles at all times throughout the simulation. 3. The map is color-coded in a continuous *rainbow* scale, ranging from blue (calm areas) over green (congested areas) to red (highly congested areas). 4. Nathalie customizes the map styling options by decreasing the opacity to 70% to be able to read the street names.
Exit Conditions	The simulation was successful and did not cause any errors.
Special Requirements	A traffic demand file must be given for the simulation.

Table 3.3: Visionary Scenario 1: Visualize Traffic Simulation Output

Chapter 4

System Design

In this chapter, the transition from the application domain to the solution domain is described on the basis of the *System Design Activities* in [BD09].

First, the *Analysis Object Model* (4.1) is built to illustrate the structure of the system. In Section 4.2, subsystems are created to decompose the system into packages. Then the dynamic behavior of the system is modeled using the UML activity diagram (4.3). To address three functional requirements, a graphical user interface is proposed in Section 4.4.

The mapping is further split into the following issues that address system design [BD09]: *Identify Design Goals* (4.5), *Identify Concurrency* (4.6), *Hardware Software Mapping* (4.7) and *Persistent Data Management* (4.8).

4.1 Analysis Object Model

The **Analysis Object Model** (AOM) describes the static structure of the system [BD09] and is shown in Figure 4.1.

The *Simulation* class has three components: the *RoadNetwork*, the *TrafficDemand* and the *SimulationOutput*. The road network and the traffic demand together represent the simulation input (2.2.3). The simulation further stores simulation settings and a simulation strategy. Traffic planners have the option to customize these settings and then start a new simulation. During a running simulation, the current step always represents the simulation progress.

The road network contains edges that describe all streets within the network as geometric edges. The network further contains the road junctions. The containing connections represent the associations between edges and junctions, or rather which edges are connected to each other through which junction. Traffic planners can use the *load*() method to import or down-

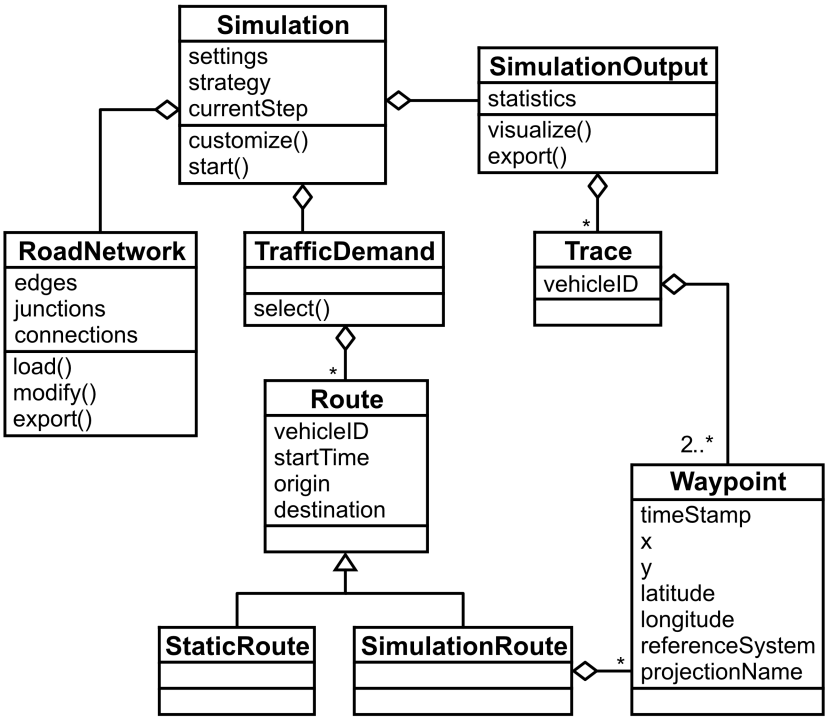


Figure 4.1: UML Analysis Object Model

load a road network. Besides, the street layout can be modified by adding new bypass roads for instance and lastly, the modified road network can be exported for permanent storage.

Traffic planners can select among static traffic demand for the morning rush hour, noon and evening rush hour. The traffic demand has many routes, whereby a *Route* either represents a *StaticRoute* or a *SimulationRoute*. Each route belongs to a specific vehicle and has a starting time, representing the time the vehicle is expected to depart within the simulation. A static route is comparable to the trip in Section 2.2.4 and only has an origin and destination location. Whereas the simulation route can have many intermediate waypoints. Therefore, a simulation route represents the path a vehicle should move along in the simulation.

When the traffic simulation is done, the simulation output can be visualized in the form of a traffic load map or exported for permanent storage. The simulation output contains simulation statistics and has many vehicle traces, whereby a *Trace* is identified by its vehicle ID and built up of at least two waypoints: the origin and destination locations. The difference compared to the simulation route is that the trace represents the actual movement of the vehicle, with one waypoint for each simulation step. Therefore, the time

stamp is needed, which is not relevant for the simulation route. Based on the time stamp, the trace indirectly includes the vehicle's driving speed or waiting time. For instance, several waypoints at the same location indicate that the vehicle was standing.

The *Waypoint* class also contains the geographical latitude and longitude coordinates for the representation of the vehicle's location in the real world and x, y coordinates for the usage within the simulation. Since coordinates without a point of origin are useless, an additional spatial reference system (2.5) is added to the waypoint class. Equally important is the map projection (2.5.2), describing the mapping type from the three-dimensional surface of the earth to the two-dimensional simulation screen.

4.1.1 Stereotypes

Moreover, all classes can be mapped to stereotypes as follows:

- **Boundary Objects**: Traffic planners interact with the *RoadNetwork* and observe the visualized *SimulationOutput*.
- **Control Objects**: The *Simulation* mediates between boundary and entity objects.
- **Entity Objects**: *TrafficDemand*, *Route*, *StaticRoute* and *SimulationRoute*, *Waypoint* and *Trace* keep persistent information.

4.2 Subsystem Decomposition

To simplify the system defined in Section 4.1, the classes are packed up into four subsystems [BD09], which are also visualized in Figure 4.2.

1. **Simulation Controller**: Contains the *Simulation* class and is responsible for the customization and execution of the simulation. The subsystem provides a simulation interface and requires the following three subsystems to get the simulation input and the simulation output.
2. **Road Network Editor:** Contains the *RoadNetwork* class and features an editing service for the modification of the road network. Besides, the export service provides the simulation controller with the road network, which is necessary as the first part of the simulation input.
3. **Route Planner**: Contains the classes: *TrafficDemand*, *Route*, *StaticRoute* and *SimulationRoute*, *Waypoint*. The route planner subsystem

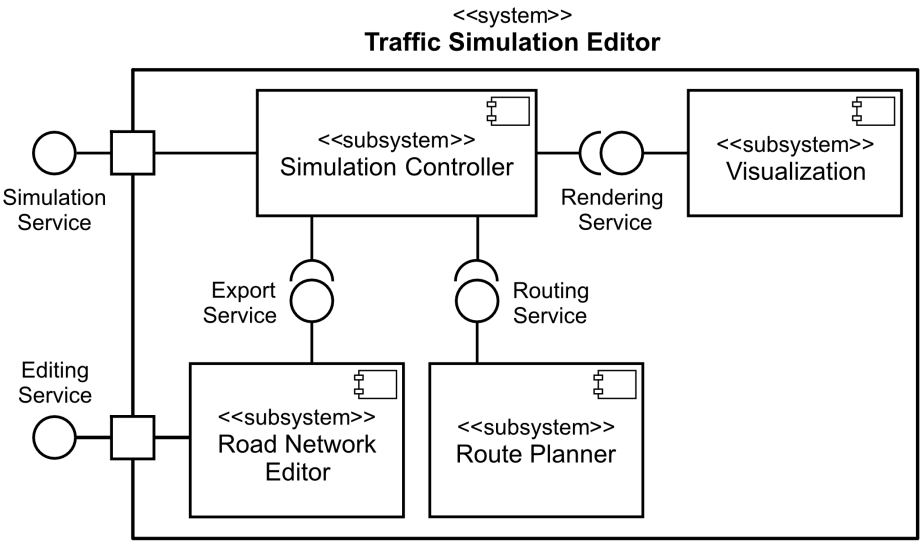


Figure 4.2: UML Subsystem Decomposition

offers a conversion service to handle the creation of simulation routes based on static routes. Thereby, the simulation controller can obtain the second part of the simulation input.

4. **Visualization**: Contains the *SimulationOutput* and the *Trace* class. The subsystem offers a rendering service to display the simulation output in the form of traffic load maps.

The waypoint class could be also added to the *Visualization* subsystem, since both trace and simulation route are associated to this class. However, the focus was put on the traffic demand and the route planning, where the waypoints play an important role to differentiate between the static routes and simulation routes.

4.3 Dynamic Behavior

The **Activity Diagram** describes the dynamic behavior of the system [BD09] and is shown in Figure 4.3.

The traffic planner opens the application by double-clicking the application icon. Then the road network can be loaded into the road network editor as a new layer. The traffic planner has the option to modify the road network as long as wanted, whereby each modification updates the road network layer by applying the changes to it. When the modification process is finished, the

simulation settings can be customized. Therefore, the traffic planner can select among different traffic demand, describing a morning, noon and evening scenario. Another option is to adjust the simulation length. After the customization part, the traffic simulation can be performed. The result of the simulation is stored in the traffic simulation output, which forms the basis for the creation of the traffic load maps. The outcome of this creation process are two visualizations: a map that provides a general overview of the traffic load and a map that allows detailed insight into the traffic load.

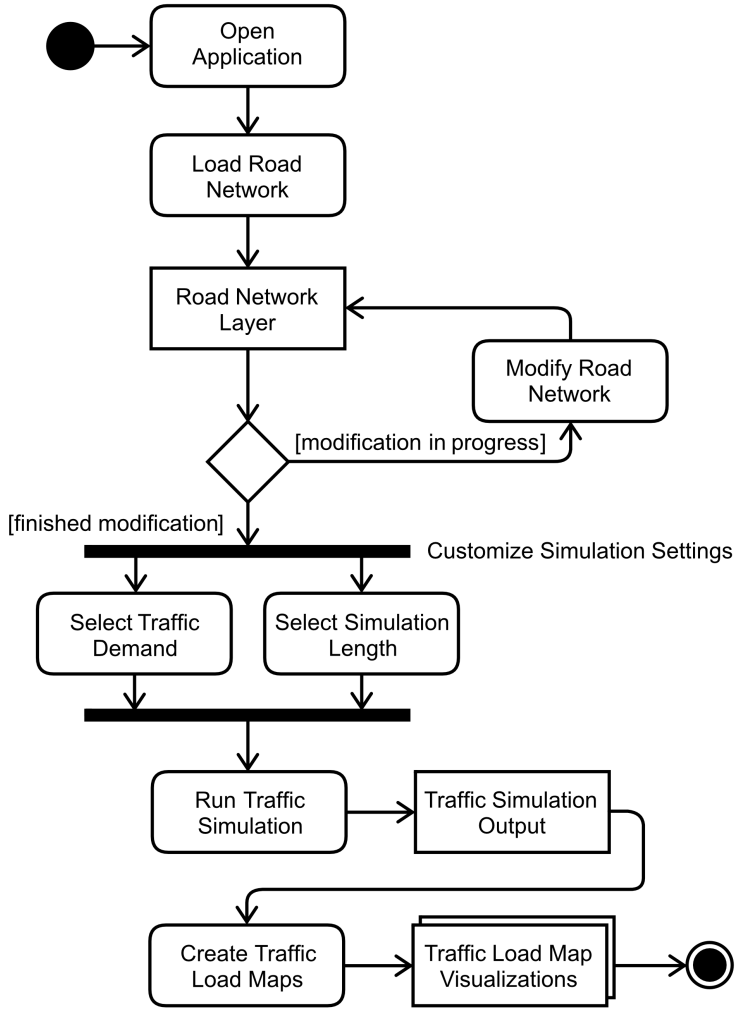


Figure 4.3: UML Activity Diagram of the System's Workflow

4.4 Graphical User Interface

For the implementation of the functional requirements [FR4], [FR5] and [FR6] (3.2), a graphical user interface (GUI) is proposed. It is illustrated in Figure 4.4. Using the GUI, traffic planners can select the traffic demand, customize the simulation settings and start the simulation.

4.4.1 Simulation Model

To save the simulation settings that are modeled as an atomic attribute of *Simulation* so far (4.1), an additional *Simulation Model* is introduced. The model includes the following attributes:

- showSimulation: *Boolean*
- simulationLength: *Integer*
- stepLength: *Double*
- trafficDemand: *String*, represents the selected traffic demand. The options are *random* or *static* traffic demand. The latter contains a sub-selection of a *morning*, *noon* and *evening* scenario.
- autoStart: *Boolean*
- saveSettings: *Boolean*
- showStatistics: *Boolean*

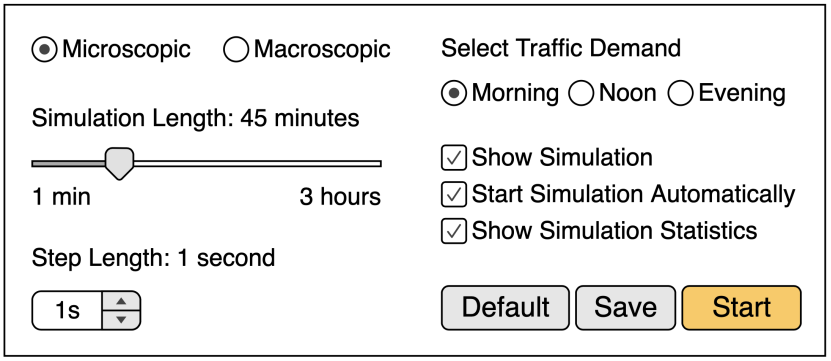


Figure 4.4: Proposed Control Panel

4.5 Design Goals

This section indicates the direction of the system's focus points.

4.5.1 Chosen Traffic Simulation

In this thesis, the focus is put on microscopic traffic simulations. The reasons for this lie in the small population of Kirchheim (around 11,000 inhabitants) and Kirchheim's focus on the individuals, trying to persuade them to switch from their own vehicles to public transport and commuter busses.

Table 4.1 illustrates some of the most popular traffic simulation software applications. The review follows the comparative analysis in [PMS13] and is extended with the results of other comparative studies such as [KH09, Zil18, ZNKE17, BBEK11]. The rows primarily contain the defined requirements.

Table 4.1 highlights that most of the traffic simulations are microscopic ones and only three of them additionally support mesoscopic simulations. However, due to the portability requirements [NFR5] of 3.3, where it is stated that the system must support both Microsoft Windows and Apple's macOS, less than half of the simulations will be considered further. Next, after applying the [NFR9] of 3.3, only freely available and open source simulations will be taken into account. These platforms are SUMO, MATSim and FreeSim. Lastly, SUMO with its extensive range of graphical features performs significantly better than MATsim (very limited GUI options, no support to create traffic scenarios) and FreeSim (does not feature a graphical simulation output). Summing up, SUMO is probably the best simulation platform that meets the defined requirements of 3.3. Besides, SUMO features an option to convert and import OpenStreetMap data, which is a huge benefit for the development process of this work.

4.5.2 Chosen OpenStreetMap Editor

There exists a variety of OpenStreetMap editors: online editors, desktop applications, mobile apps or GIS applications with additional OSM editing capabilities. However, according to the OpenStreetMap community [Ope19c] and the authors of [HW08], the most powerful editor is JOSM. So, therefore, JOSM is chosen as the editor for this thesis. It features offline editing and is platform independent due to its implementation in Java. However, as JOSM is implemented in Java, the primary language for any JOSM plugin must be also Java.

X ... supported *O ... not supported* *! ... limited support*	**SUMO**	**MATSim**	**FreeSim**	**CityMos**	**AIMSUN**	**PTV Vissim**	**PTV Visum**	**CORSIM**	**SimTraffic**	**Paramics**	**TransModeler**
Characteristics											
Microscopic	X	X	X	X	X	X	O	X	X	X	X
Mesoscopic	X	O	O	O	X	O	O	O	O	O	X
Macroscopic	O	O	X	O	X	O	X	O	O	O	X
Portability											
Microsoft Windows	X	X	X	O	X	X	X	X	X	X	X
Apple macOS	X	X	X	X	X	O	O	O	O	O	O
Linux	X	X	X	X	X	O	O	O	O	O	O
Legal											
Free	X	X	X	O	O	O	O	O	O	O	O
Open Source	X	X	X	O	O	O	O	O	O	O	O
Usability											
Graphical User Interface	X	!	!	X	X	X	X	X	X	X	X
Interfaces											
OpenStreetMap	X	X	!	!	X	X	X	O	O	O	!

Table 4.1: Review of Traffic Simulation Software

4.5.3 Trade-Offs

When design goals conflict with each other, choices must be made, which are called trade-offs. The trade-offs within this thesis are reviewed in the following section and the decisions taken are presented.

Functionality vs. Usability

One of the most common trade-offs within software engineering is the issue of multiple features and a system that is easy to use. In this case, one such trade-off is to find a proper number of configuration options for the traffic simulation.

SUMO and related tools, such as NETCONVERT or DUAROUTER, provide interfaces with dozens of additional configuration options. These options are useful for experienced traffic planners, however they might be redundant for smaller communities. Therefore, the decision was made to predefine simulation options that are related to common simulation tasks and only enable the modification of a specific subset of simulation options.

These predefined options include the *Join Junctions* feature within NETCONVERT, which is recommended for the use of OpenStreetMap data, and several *Repair* and *Depart and Arrival* options within DUAROUTER to correct invalid routes and origin/destination positions of the vehicles.

The options that are open to traffic planners include the simulation mode, simulation demand, simulation output, simulation length and simulation step length.

Rapid Development vs. Functionality

Another common trade-off is the implementation of various features within a limited time frame. Therefore, the functions of 3.2 were ordered by priority and each iteration of the development process covered the highest ranked feature first. As a result, the additional simulation statistics [FR9] are still not implemented.

The simulation settings control panel (4.4) illustrates another issue of the rapid development/functionality trade-off. Currently, the number of simulation settings is limited, as described in Section 4.5.3. Most notably, the selection between *Static (morning, noon, afternoon)* and *Random* traffic demand is modeled primitively. Traffic planners can either choose static traffic demand for the municipality of Kirchheim or random traffic demand that covers the whole world. The decision to model the traffic demand selection in such a way is based on the importance of realistic traffic demand data for Kirchheim exclusively, since the municipality is the first possible end-user of

the developed system. Only the static traffic demand allows traffic planners to compare simulation results with meaningful outcomes. Simulations based on random traffic demand data always produce diverse results, even though the road network has never changed.

Consequently, the visionary scenarios in Section 7.3 suggest a future implementation that covers extended functionality for the creation and selection of persistent traffic demand data, applicable to the entire world.

Efficiency vs. Portability

The nonfunctional requirement [NFR8] from 3.3 was mapped to the use of the *Docker* engine (more details are covered in Section 4.7.1). Thereby, the focus was placed on a simplified integration process between the traffic simulation and the road network editor in addition to an easy distribution of the system. However, a notable disadvantage of using Docker is the delayed operability during zooming and the fact that some of the graphical features are displayed skewed in JOSM.

4.6 Identify Concurrency

The goal is now to identify objects that are inherently concurrent in order to achieve performance relevant nonfunctional requirements. For each concurrent object, a unique thread can be implemented. To figure out threads for concurrent tasks, the *concurrency questions* of [BD09] are applied.

First of all, if the system does not support multiple users and only one control object exists, then we do not need threads. Second, entity objects that can be executed in parallel are analyzed. Several entity objects were derived in Section 4.1.1 and decomposed into two subsystems (4.2). Since both subsystems might contain waypoints, none of the entity objects can be executed simultaneously.

The last question refers to requests that might be split into multiple parts and handled in parallel. The only requests that will be relevant in the system are conversion tasks during the preprocessing of simulation input data and the post processing of simulation output. These processes are strictly sequential because they are based on files (see Section 4.8), where a file can only be appropriately read by the next processing tool after it is closed by the current one.

In addition, the traffic simulation SUMO does only support mulitple threads for the route calculation of the vehicles. Per default, the number of threads for the routing is set to one but can be increased. The SUMO

tool DUAROUTER (2.2.4) features exactly the same parallelization options [Ger19]. However, the traffic simulation itself is always single-threaded. This could lead to time-consuming computations if the simulation length is several hours and hundreds to thousands vehicles are included [Pot12, KHRW02].

4.7 Hardware/Software Mapping

Associations between subsystems are mapped to network connections [BD09]. The *Simulation Service* can be realized with a simple client-server architecture, where the simulation process itself represents the server and the simulation controller represents the client. The client always increments and sends the current step to the server. Then the server executes the next simulation step, notifies the client about the current status and waits again for the client-request.

The *Routing Service* and the *Rendering Service* can be implemented with a client-server architecture as well. Both the visualization subsystem and the route planner subsystem can operate as servers and therefore handle application layer[1] requests that tell the servers what to do.

The services provided by the road network editor are not suitable for a network connection mapping. The *Export Service* only saves the road network layer as a file (see Section 4.8) and the *Editing Service* presents the graphical interface to the user of the system, probably the traffic planner.

4.7.1 Container Platform

In order to realize the nonfunctional requirement [NFR8] from 3.3, the decision was made to use the *Docker* engine. Docker turns the system into a portable system that can be distributed easily to several machines and platforms [Tur19]. Therefore, a so-called Docker *image* is created. It consists of several layers that represent Docker commands. The running instance of a Docker image is called Docker *container*. A Docker container takes up more space than a local installation would need (by a factor of ten for JOSM) but it is still lightweight compared to virtual machines.

Based on the decomposed subsystems (4.2), two Docker containers are proposed. One container should primarily include the road network editor and the visualization subsystem while the other container includes the traffic simulation and the route planner.

[1] Application layer = layer 7 of the OSI (Open Systems Interconnection) model

4.8 Persistent Data Management

In Section 4.1.1, entity objects were identified. These objects have to be persistent to store the values of their attributes permanently [BD09]. For this thesis, the decision was made to follow the *file system* mechanism for persistent storage.

The road network editor is the only writer of road network files and the traffic simulation software is the only writer of the simulation output. Therefore, there is no need to support multiple writers for the same file as could be done with a database mechanism. Furthermore, the chosen traffic simulation software SUMO relies on input data in the form of files [BBEK11].

Chapter 5

Agile City Planning Suite

The Agile City Planning Suite (ACPS)[1] is the actual implementation of the system designed in Chapter 4. It consists of three major components:

1. **JOSM Container**: Includes JOSM and the **TraLAMA** (Traffic LoAd MAp) **Plugin** for JOSM.
2. **SUMO Container**: Includes SUMO, the *Route Planner* subsystem and a RESTful[2] API[3] to enable communication with them from the JOSM container.
3. **TraLAMA Web Service**: Includes another RESTful API for the visualization of traffic load maps.

ACPS uses the REST paradigm for the communication between the three components. Besides, Docker serves as a main hub between the two containers. They are connected using a Docker bridge and share a common volume for the exchange of data. Both containers are based on the operating system Linux Ubuntu.

5.1 Overview

In Section 5.2, new solution objects are identified to refine the analysis object model. Then the functionalities of each new object are described. In the subsequent Section 5.3, the overall features of the JOSM container are presented. Section 5.4 introduces the implemented TraLAMA plugin and its

[1] ACPS is accessible via `https://repobruegge.in.tum.de/scm/dsm/jakobs.git`
[2] REST (REpresentational State Transfer)
[3] API (Application Programming Interface)

functionalities. The section further provides insight into the communication process with the SUMO container and finishes with a workflow description. Then the features of the SUMO container are presented in Section 5.5. In particular, the focus is on the *SimulationController*, the network connection to the traffic simulation SUMO and two route planner tools which are responsible for the preprocessing of the simulation input data. Section 5.6 is all about the implemented TraLAMA web service and the provided communication point. Lastly, Section 5.7 explains the interaction with TraLAMA from the perspective of traffic planners.

5.2 Object Design

In this section, the developed object model (4.1) and the decomposed subsystems (4.2) are refined. Therefore, additional solution objects must be identified [BD09]. The resulting solution object model is illustrated in Figure 5.1.

The *RoadNetwork* object, the only subset of the *Road Network Editor* subsystem, can be directly mapped onto the existing system JOSM. Since JOSM allows traffic planners to modify and export OpenStreetMap data, the *Editing Service* and the *Export Service* of the road network editor subsystem are already provided.

The entire *Visualization* subsystem, including the *SimulationOutput* object and the *Trace* object, were originally mapped to JOSM as well. However, the styling options were very limited and therefore the decision was made to realize the visualization subsystem and its *Rendering Service* with an additional web service, which will be introduced in detail in Section 5.6.

In Section 4.4, a graphical control panel and a related simulation model were introduced. For the realization of these objects, the JOSM plugin **TraLAMA** is developed. TraLAMA directly extends the object model with two new classes: *TraLAMA* and *Plugin*. TraLAMA represents the entry point of the JOSM plugin and therefore inherits from the *Plugin* class that is provided by JOSM. The *Plugin* class is drawn in a lighter gray because it is used without any modifications. Given the TraLAMA plugin, the afore mentioned control panel and the simulation model can be added to the solution object model as *SettingsView* and *SimulationModel*. The *SettingsView* further provides elements to start the traffic simulation (4.4) and hence, the *SimulationService* can also be mapped to this object.

The *Simulation* class, which represents the *SimulationController* subsystem, is refined into three components: *SimulationThread*, *SimulationInterface* and *SimulationController*. That is, because the traffic simulation SUMO

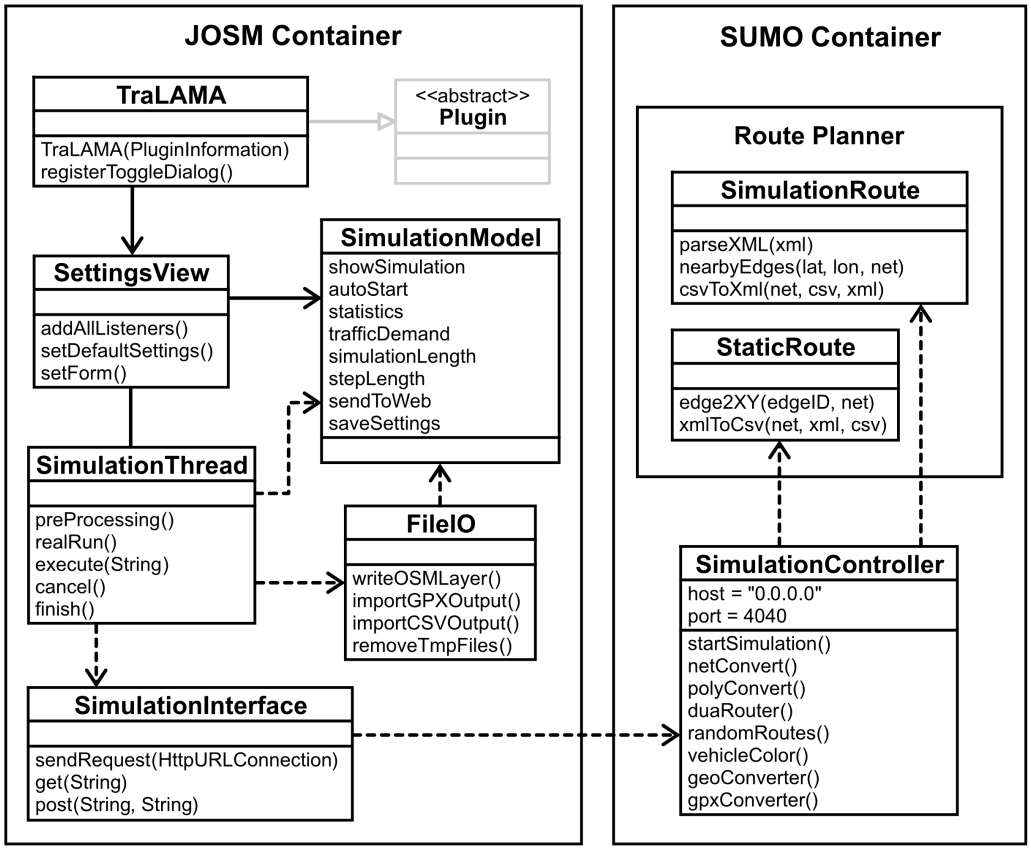


Figure 5.1: TraLAMA Object Model

is deployed on a different container than the editor JOSM and TraLAMA (see Section 4.7). Therefore, the simulation thread creates a new worker thread that prepares all necessary commands used for the preprocessing of the simulation input data, the simulation itself and the post processing of the simulation output. The simulation thread object passes these commands to the simulation interface, which wraps them up into network packages. These network packages are then sent to the SUMO container, where the simulation controller handles the execution of the simulation relevant tasks (detailed description in Section 5.4.2) and stores the results in files (as defined in Section 4.8). To access these simulation result files, *FileIO* is introduced.

The entire *Route Planner* subsystem is mapped onto the SUMO container, since all included objects require features provided by the traffic simulation. Supplementary tools were added to the route planner subsystem to preprocess the simulation input data and post process the simulation output. The preprocessing tools can be further mapped to the *Routing Service* of the

original route planner subsystem.

5.2.1 Stereotypes

The resulting classes of the solution object model can be assigned with stereotypes:

- **Boundary Objects**: Traffic planners interact with the *SettingsView*.
- **Control Objects**: *TraLAMA*, *SimulationThread*, *SimulationInterface* and *SimulationController* mediate between boundary and entity objects.
- **Entity Objects**: *FileIO*, *SimulationModel* and all classes within the *Route Planner* subsystem keep persistent information.

5.2.2 Object Description

The classes feature the following functionalities:

- **TraLAMA**: The TraLAMA class needs a constructor that will be invoked by JOSM. Besides, the graphical control panel (in the form of a *ToggleDialog*) needs to be registered.
- **SettingsView**: This class represents the graphical control panel. First of all, listeners are assigned to the control elements to observe and handle performed actions. Based on previously stored simulation settings, the graphical elements are adjusted. When the traffic planner presses the *Default* button, the simulation settings are set to default.
- **SimulationModel**: The simulation model was already introduced in Section 4.4.1. It is now extended with the **sendToWeb**:*Boolean* attribute because an additional web service is added to the system (see Section 5.6).
- **SimulationThread**: The actual worker thread first performs two preprocessing steps: removing temporary files as well as filtering and writing the OSM layer using *FileIO*. The *realRun*() method represents the JOSM version of the Java *run*()[4] method and executes all relevant tasks within a new thread. The *execute*(*String*) method passes a new HTTP (Hypertext Transfer Protocol) request to the *SimulationInterface* and

[4] `https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html`

updates the progress bar, which shows the current task of the processing pipeline. After the thread is finished, the *finish*() method will be invoked and the user gets notified with a textual message about the simulation result. Furthermore, a new thread can always be canceled.

- **FileIO**: Using FileIO, the currently selected OSM layer can be filtered for unnecessary elements and saved to a file. Another feature of this class is to import the GPX (GPS Exchange Format) formatted simulation output and visualize it within JOSM. The CSV (Comma-Separated Values) formatted output can be imported as well and sent to the TraLAMA web service. Lastly, FileIO provides a method to remove all temporary files within a folder. The files which are usually removed in TraLAMA include simulation input and output files of previously performed traffic simulations.

- **SimulationInterface**: This class features the networking capabilities within TraLAMA. Therefore, a new HTTP connection can be set up to send HTTP GET and POST requests. The *get*(*String*) method requires a String-URL (Uniform Resource Locator) as a parameter, whereas the *post*(*String*, *String*) method requires both an URL and an additional JSON (JavaScript Object Notation) body.

- **SimulationController**: The simulation controller represents a web server that is listening to all public IP addresses, incoming at port 4040. The controller is able to start a new SUMO simulation and perform NETCONVERT (2.2.4), POLYCONVERT (2.2.4) and DUAROUTER (2.2.4). Additionally, random SUMO routes can be generated and vehicles within a SUMO simulation can be colored. Using the *geoConverter*() method, the simulation output can be converted to the GPX format, representing the traces of the vehicles. The GPX file can be further parsed into a CSV file using the *gpxConverter*() method, though the information about the individual vehicle gets lost during this process.

- **StaticRoute**: Is a tool that can convert SUMO trips into a CSV file with geo-coordinates. Details are covered in Section 5.5.2.

- **SimulationRoute**: Another tool that converts static routes with geo-coordinates into simulation routes for SUMO. Details are covered in Section 5.5.2.

Figure 5.2: Modified OSM Road Network in JOSM

5.3 JOSM Container

The JOSM container includes the newest tested and built JOSM executable and the TraLAMA plugin. The TraLAMA plugin is connected to JOSM by inheriting the provided *Plugin* class of JOSM, which is the recommended way of integration.

In Figure 5.2, an extract of JOSM is shown that contains a newly added street called *TUM Exzellenzstraße*. It is connected to the *Heimstettener Moosweg* with a roundabout and to the horizontal street with a traffic light. JOSM already provides the functionality of adding, modifying or removing streets and junctions. The aerial imagery used for the background is derived from the Bavarian Survey Authority (German: *Bayerische Vermessungsverwaltung*) and can be easily imported in JOSM.

5.4 TraLAMA Plugin

TraLAMA is a plugin for JOSM that establishes a bridge between the two existing systems SUMO and JOSM. It allows traffic planners to export modified OpenStreetMap data, run a traffic simulation in SUMO and view the

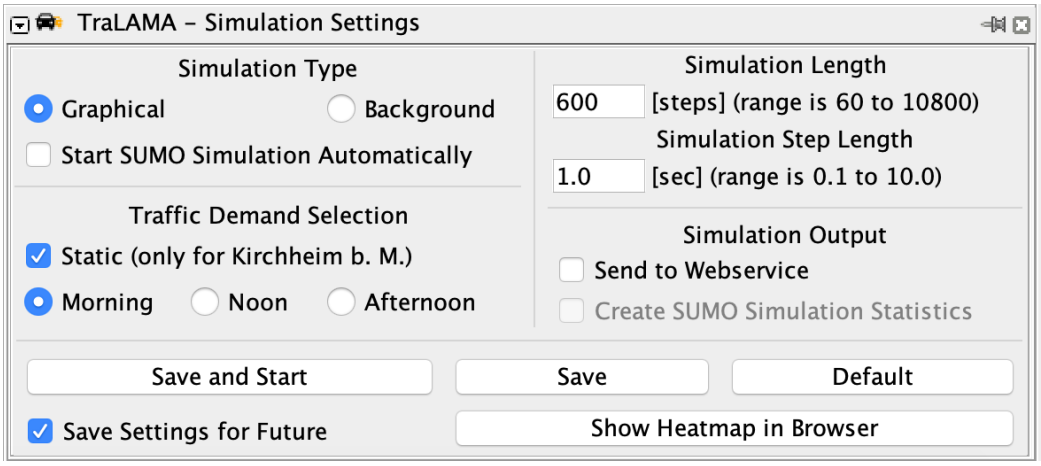


Figure 5.3: TraLAMA Control Panel

results in JOSM as additional traffic load map layers. To make it easier for traffic planners, TraLAMA implements a resizable graphical control panel with selectable GUI elements to modify the traffic simulation settings. Such a panel within JOSM should inherit the *ToggleDialog* class. In doing so, the settings panel can be activated and deactivated. In JOSM, all activated graphical control panels are displayed in a row, right next to the map view.

Figure 5.3 shows the activated control panel of the TraLAMA plugin in JOSM. It can be used to customize the settings of the traffic simulation and to start a new simulation. The *User Interface* (UI) is based on the introduced control panel in Section 4.4.

TraLAMA is implemented in such a way that a new traffic simulation can only be started if an OSM layer is selected. Besides, the simulation settings within the control panel are stored permanently using *Java Preferences*. It is further ensured that only one traffic simulation is executed at the same time.

5.4.1 Filter OSM Elements

The *FileIO* class implements the method *writeOSMLayer*(). This method stores the currently selected OSM layer in a file that can be accessed by the traffic simulation SUMO. Before the layer is saved, it is filtered for key-value pairs such as *highway=footway* or *vehicle=no*. These keywords emerged as useful throughout the evaluation (Chapter 6) and remove roads from the road network which should be inaccessible to the simulated vehicles.

5.4.2 Communication with SUMO Container

TraLAMA runs within the JOSM container, which is connected to the SUMO container using a Docker bridge. Since Docker also acts as a nameserver, the IP (Internet Protocol) addresses of the containers are automatically translated into the specified names of their provided services. In ACPS, these services are called *josm* and *sumo*, for the JOSM and SUMO container respectively. Concurrently, each Docker container exposes a unique port. The combination of the service name and its port together build the interface of the service. For instance, the service *sumo*, exposed at the port 4040, is reachable via [5].

The connection between TraLAMA and the SUMO container works as follows: the *SimulationInterface* sends HTTP requests to the *SimulationController* and receives response messages. The GET requests as well as the POST requests are encoded with the the URL [6], whereby the *command* key must be either *sumo*, *statictrips* or *randomroutes* for POST requests, or *netconvert*, *polyconvert*, *duarouter*, *vehiclecolor*, *geoconverter*, *gpxconverter* or *xmlconverter* for GET requests. All mentioned commands are covered in detail in Section 5.5.1.

Each POST request is extended with a JSON content body. Inside this body, additional information is stored in the form of *key-value* pairs. The *keys* are predefined and based on the *Simulation Model* in Section 4.4.1, whereas the *values* are derived from the currently selected simulation settings. Examples for the JSON contents are illustrated in Listing 5.1, Listing 5.2 and Listing 5.3.

Listing 5.1: Exemplary JSON Body of the Command ***statictrips***

```
{
  "trafficDemand": "morning",
  "simulationLength": 600
}
```

Listing 5.2: Exemplary JSON Body of the Command ***randomroutes***

```
{
  "simulationLength": 600,
  "stepLength": 1.0
}
```

[5] `http://sumo:4040/`
[6] `http://sumo:4040/command`

Listing 5.3: Exemplary JSON Body of the Command ***sumo***

```
{
  "showSimulation": "True",
  "simulationLength": 600,
  "stepLength": 1.0,
  "autoStart": "True",
  "showStatistics": "False"
}
```

5.4.3 Generation of Traffic Load Maps

The simulation output of SUMO contains the traces of all vehicles. Using the *GeoConverter* tool of the route planner subsystem, the simulation result can be converted into a GPX file. TraLAMA prints this GPX file as new *traffic load map layer* in JOSM. Therefore, the default JOSM preferences were adjusted in such a way that the SUMO simulation output is always rendered in the style of a heat map.

The simulation result is further sent to the TraLAMA web service using the HTTP POST request. The JSON body of this request contains the traffic simulation settings and an array of geo-coordinates (latitude, longitude) which represents the traces of the vehicles. Using the TraLAMA web service, the traffic load map is displayed with additional configurable styling options (see Section 5.6).

5.4.4 TraLAMA Workflow

As discussed in Section 4.6, TraLAMA operates strictly sequentially because each processing step depends on a previous step. With regard to the network connection with the SUMO container (5.4.2), concurrent requests are not allowed. This means that after a request is passed from the *SimulationInterface* to the *SimulationController*, the interface waits for the response and does not send further requests during this time. If the controller reports success, the next request can be sent. Otherwise, TraLAMA immediately stops the current *SimulationThread*.

The TraLAMA workflow can be divided into three parts: Preprocessing, where the simulation input is prepared for the simulation, the traffic simulation itself and post processing, where the simulation results are visualized.

Preprocessing

When the traffic planner presses the *Save and Start* button, the currently selected simulation settings within the control panel are saved and a new *SimulationThread* is created. This thread performs a number of functions.

First of all, the thread removes all files within the *output-simulation* folder where the simulation results will be stored. This is to ensure that no file conflicts appear. Then the currently selected OSM layer is filtered (5.4.1) and stored in a new *OSM file*. These processing steps represent the foundation on which the *SimulationController* of the SUMO container can then operate.

In the next step, the *SimulationThread* prepares the required URL commands and JSON content bodies as described in Section 5.4.2. First, a GET request for *netconvert* is created and sent to the *SimulationController* to create a road network. Second, the *polyconvert* GET request is passed to the controller to create additional geometrical shapes, but only if the traffic planner selected the graphical simulation type. Depending on the selection of the traffic demand, different requests are then necessary. For static traffic demand, the POST request *statictrips* with the JSON body of Listing 5.1 is sent followed by the GET request *duarouter*. In doing so, simulation routes are created based on the static traffic demand. If the traffic planner had chosen random traffic demand, a different POST request with the command *randomroutes* and the JSON body of Listing 5.2 would be needed. This creates random simulation routes. Lastly, the *vehiclecolor* GET request is sent to the *SimulationController* to color the vehicles based on their location of origin.

Traffic Simulation

Then the SUMO traffic simulation can be started by passing a POST request with *sumo* and the JSON body of Listing 5.3 to the *SimulationController*.

Post Processing

After the simulation is finished, the simulation result needs to be converted before it can be displayed in the form of a traffic load map. Initially, the *geoconverter* GET request is sent to the controller to convert the simulation output into a *GPX file*. Then the *SimulationThread* is able to import this result file using the *importGPXOutput*() method within *FileIO*. If the traffic planner ticked the *Send Simulation Output to Web Server* option, another *gpxconverter* GET request is created and sent in order to translate the GPX file into the CSV format. Then the *SimulationThread* can parse the resulting *CSV file* into a JSON object and transmit it to the **TraLAMA Web**

Service. If the server reports success, the thread notifies the traffic planner about the successful simulation processes. In the event that TraLAMA stopped the *SimulationThread* due to an error, the traffic planner is updated with a different notification, indicating premature termination.

5.5 SUMO Container

The SUMO container runs the built SUMO traffic simulation software and all its supplemental tools and libraries such as NETCONVERT (2.2.4), POLYCONVERT (2.2.4), DUAROUTER (2.2.4) and TraCI (Traffic Control Interface). TraCI provides a communication point for the setting and receiving of data and variables between the SUMO simulation and any registered client, such as the *SimulationController* in this case.

Figure 5.4 illustrates a running traffic simulation based on a modified OSM network that was converted using NETCONVERT. Notable is that the vehicles are driving along the newly created road in spite of the larger distance. The reason for this is that the maximum allowed speed for this new road was increased while the speed limit for the original road was lowered.

5.5.1 Simulation Controller

In fact, TraLAMA never directly communicates with the SUMO simulation, but with the RESTful API, the *SimulationController*. Since the RESTful GET interfaces are primitive, TraLAMA can interact with the simulation controller by just sending a request containing the name of the service to be executed (see Section 5.4.2). This is possible because most of the services do not require customized parameters for each request. Thus, we can predefine executable commands as well as constants with path- and filenames. These files are all stored and accessible on the shared volume.

Predefined Commands

This section covers all predefined executable commands and their functionalities. The *SimulationController* expects additional JSON content bodies (see Listings 5.1, 5.2 and 5.3) for commands that represent HTTP POST requests. All commands are either associated with existing SUMO tools or with newly implemented tools.

1. **xmlconverter**: Executes the *StaticRoute* (see Section 5.5.2) tool to convert a SUMO trip data set that uses road network edge IDs into a

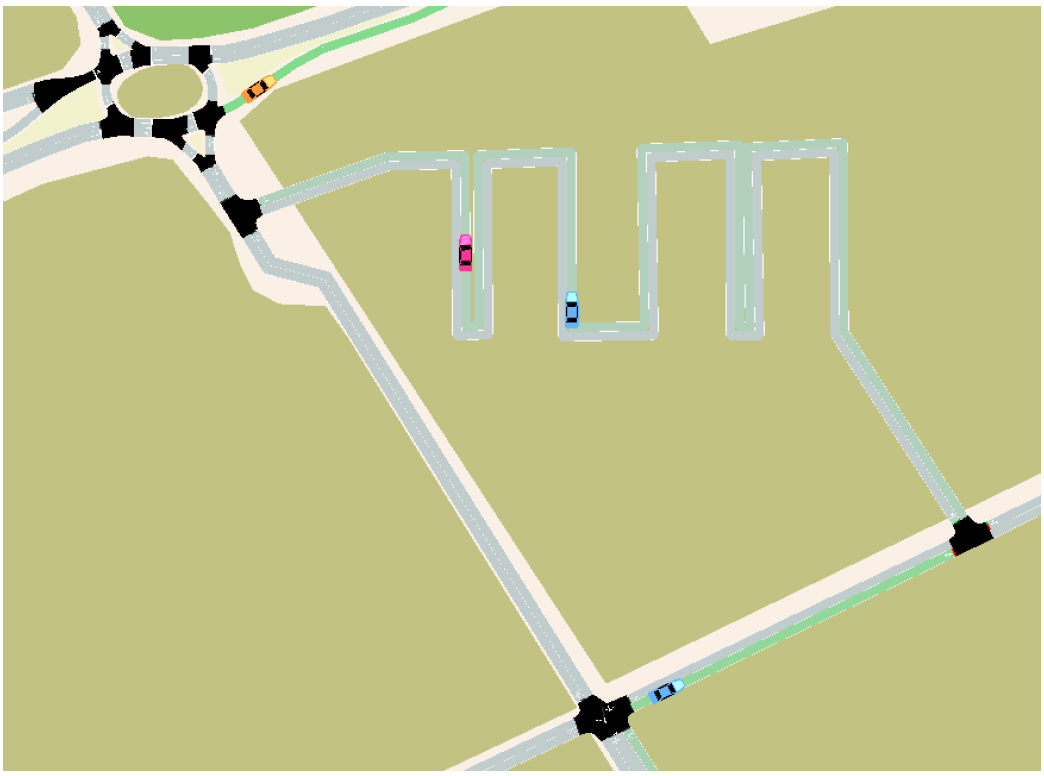


Figure 5.4: SUMO Simulation Based on a Modified OSM Road Network

CSV file with geo-coordinates. This tool was a prerequisite to generate the static traffic demand data for Kirchheim.

2. **netconvert**: Invokes the SUMO tool NETCONVERT to parse the OSM road network which was saved by TraLAMA into a SUMO road network. Additional parameters are added to join OSM junctions and include the street names in the output file.

3. **polyconvert**: Executes the SUMO tool POLYCONVERT to parse OSM points of interest (POI) and additional geometric shapes such as farmlands or buildings for the visualization in SUMO.

4. **statictrips (POST)**: Executes the *SimulationRoute* (see Section 5.5.2) tool to convert static routes with geo-coordinates into simulation routes for SUMO. Therefore, the road network file is required, which was generated by NETCONVERT. Besides, the *trafficDemand* and *simulationLength* values from the JSON content body (Listing 5.1) are needed to select the proper traffic demand scenario. The tool returns the *begin*

and *end* time of the simulation route and the *SimulationController* stores them for the SUMO simulation.

5. **duarouter**: Calls the SUMO tool DUAROUTER to calculate routes of vehicles based on the trip file that was generated by *SimulationRoute* and the road network file from NETCONVERT.

6. **randomroutes (POST)**: Executes a SUMO tool that generates random trips based on the road network from NETCONVERT. Afterwards, DUAROUTER is called to turn these trips into simulation routes.

7. **vehiclecolor**: Invokes a tool to color the simulation vehicles based on their origin. Therefore, the tool is provided with information about *Traffic Assignment Zones* (TAZ), where predefined zones in the area of Kirchheim have assigned colors. Vehicles departing in Kirchheim are colored orange, those departing from Heimstetten are colored pink and all other vehicles are visualized in blue.

8. **sumo (POST)**: Performs a new traffic simulation in SUMO. Therefore, TraCI is used to communicate with the simulation, as illustrated in Listing 5.4. The *SimulationController* represents the client and SUMO acts as the server. Initially, the TraCI library needs to be loaded. Then a new network connection is established with the *sumoCommand* as one parameter and a *port* for the server as another parameter. The sumo command is an array that consists of a *sumo binary*, which can be either *sumo* for a background simulation or *sumo-gui* to start a graphical simulation, and additional simulation settings. These settings and the sumo binary are derived from the JSON content body (Listing 5.3) and extended with the *begin* and *end* value of *statictrips*. As long as the current simulation step is less than the *simulationLength*, which is specified in the JSON content body as well, TraCI performs another SUMO simulation step. Then the current step is incremented. If an error occurs within SUMO, the *SimulationController* stops the simulation and returns -1. After the simulation is finished successfully, the connection is closed. Thereby, the *False* parameter ensures that the simulation is closed definitely, even though some vehicles might not have reached their destination yet.

9. **geoconverter**: Executes a modified version of the SUMO tool *TraceExporter* to convert the *floating car data* simulation output, which is used within this work, into a GPX track. The difference to the original

tool is that another time format is used, because SUMO simulations are based on seconds only. Therefore, the simulation time is extended with the simulation date and together they are formatted into a human readable date format which is in accordance with the ISO standard 8601[7].

10. **gpxconverter**: This command invokes another tool that converts GPX tracks into a CSV file. Thereby, information about the individual vehicle gets lost and all traces are merged into one.

Listing 5.4: TraCI Communication Between the *SimulationController* and SUMO When Using the ***sumo*** Command

```
1  import traci
2  traci.start(sumoCommand, 4041)
3  try:
4          step = 0
5          while step < int(payload["simulationLength"]):
6          traci.simulationStep()
7          step += 1
8  except:
9          return "-1"
10 traci.close(False)
```

5.5.2 Route Planner

The *Route Planner* subsystem consists of many tools for the preprocessing of simulation input data and the post processing of simulation output data. Each tool provides a command-line interface that takes at least input and export file names as parameters. This section provides deeper insight into the two preprocessing tools *StaticRoute* and *SimulationRoute*.

Create Static Routes

To analyze the impact of road network editing on traffic flow, static traffic demand is required. To create such a traffic demand automatically, the *StaticRoute* tool was implemented. *StaticRoute* takes SUMO trips as input data and creates a CSV file with geo-coordinates as output data.

Within this thesis, three different input files were used, each representing a different traffic scenario, morning rush hour, noon and evening rush hour.

[7]ISO Standard 8601 Example: 2019-08-06T04:00:01Z

Static Traffic Demand				
StartTime	**OrigLat**	**OrigLon**	**DestLat**	**DestLon**
07:42:03	48.174750	11.745215	48.171622	11.746312

Table 5.1: Exemplary CSV Entry for a Static Traffic Demand Trip

The SUMO trips file was generated with DUAROUTER, whereby additional parameters were added. These parameters primarily include adapted probabilities that are based on commuter statistics of Kirchheim, Kirchheim's population density and company locations in Kirchheim, together with the number of employees at each company. Additionally, the *validate* option was used to generate trips where a route can be found. This means that not only are trips generated, but also routes are created based on these trips. All valid routes, in other words routes that were accepted by the traffic simulation SUMO, are then converted back to trips again.

Based on the created SUMO trips files, *StaticRoute* identifies geographic coordinates for all origin and destination edges within each trips file. Therefore, the *xmlToCsv*(*net*, *xml*, *csv*) method needs to extract the origin and destination edges of each trip within the *xml* file and pass their IDs to the *edge2XY*(*edgeID*, *net*) method. Here, *net* represents the road network object which is imported using the SUMO module *sumolib*. This module provides functions to retrieve nodes and edges of the imported road network. First, *StaticRoute* fetches the edge object based on the provided edge ID. Second, the starting node of this edge is fetched and its SUMO coordinates are extracted.

Then the retrieved SUMO coordinates for both the origin and destination edge can be converted to geographic coordinates. For this coordinate transformation, *sumolib* provides the *convertXY2LonLat*(x, y) function that returns latitude and longitude objects.

The starting time of the vehicle can also be extracted from the current SUMO trip. Given all these extracted values, a new CSV row that contains the origin latitude, origin longitude, destination latitude, destination longitude and the starting time is added to the *csv* file. This process is repeated for each trip entry within the SUMO trips file. An example of a generated CSV row is illustrated in Table 5.1.

Create Simulation Routes

Before a new traffic simulation can be started, the simulation input needs to be preprocessed. While NETCONVERT prepares the road network, the *SimulationRoute* tool is responsible for the preprocessing of the static traffic demand. The tool parses static trips with geographic coordinates into simulation trips for SUMO. This means that *SimulationRoute* features exactly the opposite functionality of *StaticRoute* (5.5.2).

Using the *csvToXml*(*net*, *csv*, *xml*) method, the tool iterates over the *csv* file line by line and computes the nearest edges to the given origin and destination geo-coordinates for each row. Therefore, the *nearbyEdges*(*lat*, *lon*, *net*) method is called twice, for both the origin and destination location. First, this method converts the latitude and longitude parameters to SUMO coordinates. This coordinate transformation is enabled through the SUMO module *sumolib* and its *convertLonLat2XY*(*lon*, *lat*) function. Second, all neighboring edges within a specified radius are identified, using the *sumolib* feature *getNeighboringEdges*(*x*, *y*, *radius*). Third, the *SimulationRoute* sorts the edges and picks the closest one.

In the next step, the edge IDs are derived for the origin and destination edges using the *parseXML*(*xml*) method. Lastly, the *SimulationRoute* tool creates a new XML trip entry including an ID, the departure time, origin and destination edge IDs and appends this entry to the *xml* file. The ID starts with 0 and is incremented until it is equal to *simulationLength*, which is a general input parameter for this tool. The starting time can be directly derived from the CSV row without any modifications. An example of such an entry is illustrated in Listing 5.5. The departure time must be given in seconds for the simulation with SUMO and represents the time *07:42:03*. The process of creating and appending new entries to the resulting *xml* file is repeated for each trip entry within the CSV trips file.

After the entire trips are parsed, *SimulationRoute* returns the first and the last starting time values. This information is required for the *SimulationController* to start the SUMO simulation within a time frame for which traffic demand exists.

Listing 5.5: Exemplary XML Entry for a SUMO Trip

```
<trip
  id="23" depart="27723.0"
  from="394892651" to="52937834"
/>
```

5.6 TraLAMA Web Service

The TraLAMA web service provides a HTTP POST interface for TraLAMA and a HTTP GET interface for traffic planners.

The POST interface expects and only accepts HTTP requests with a JSON content body. This content body must contain information about the traffic simulation settings and an array of geo-coordinates (latitude, longitude). The array describes the traces of all vehicles throughout the simulation, which is equivalent to the locations of each vehicle at all times. When a POST request is registered, the web service splits the settings information from the geo-coordinates and stores them into two separate files. Listing 5.6 shows an appropriate content body for such a POST request.

Listing 5.6: Exemplary JSON Body for the POST Request to the TraLAMA Web Service

```
{
  "user": "TraLAMA",
  "date": "2019-09-09",
  "trafficDemand": "morning",
  "simulationLength": 1800,
  "stepLength": 1,
  "simulationOutput": [
    {
      "lat": 48.171548,
      "lon": 11.754922
    },
    {
      "lat": 48.171569,
      "lon": 11.754916
    }
  ]
}
```

The GET interface requires the afore mentioned files (settings information, geo-coordinates) to display a traffic load map and some relevant information about the simulation settings. In addition, a control panel is shown to allow traffic planners to customize the styling options of the map. The first styling option is to change the threshold of the coloring scheme. By increasing the value of the threshold, the focus of the map is moved to highly congested areas. The next styling option is to change the radius for the thickness of the displayed vehicle tracks. Using the blur option, traffic planners can adjust the coloring to a more discrete scheme. In doing so, congested

Figure 5.5: Traffic Load Map in the Web View

parts of junctions can be easier identified. Lastly, traffic planners are able to reduce the opacity for the displayed map to increase a better visibility of the underlying street names, particularly in highly congested areas.

Figure 5.5 shows an example of a traffic load map that is displayed on the TraLAMA web page. It is notable that the traffic load map spreads out over the cropland because the web page uses realistic OSM data whereas the simulation was based on modified OSM data.

5.7 User Workflow

This section presents the interaction between a traffic planner and ACPS. The usual workflow repeats the steps two to five to get a comparison between different road network models.

1. **Starting and Loading OSM Data**: The traffic planner starts the *Agile City Planning Suite* with a double-click on the TraLAMA application (macOS) or using Lifeboat[8] (Windows). This starts both the JOSM container and the SUMO container. JOSM and the activated TraLAMA plugin are rendered in a graphical panel for the interaction with the traffic planner while the SUMO container only prepares the *SimulationController* to listen for incoming TraLAMA requests. Then

[8] `https://uselifeboat.com`

the traffic planner can import OSM data from his local drive or via the web API of JOSM, as illustrated in Figure 5.6. Moreover, the traffic planner can add aerial imagery.

2. **Modify and Validate OSM Data**: JOSM provides various tools for the traffic planner to modify OSM data. For instance, the traffic planner can add new lanes, exchange traffic lights with roundabouts and remove unnecessary roads. Furthermore, JOSM includes a control panel for the traffic planner to validate OSM objects. The data validation check is used to ensure that the OSM elements are tagged appropriately. For example, ways must have a positive number of lanes and unnecessary tags such as duplicated or implicit ones (*foot=yes* is useless when the element is already tagged with *highway=footway*) should be avoided. If errors occur, the traffic planner can try to fix them before proceeding to the next step. Figure 5.7 shows the validation control panel and the tag panel of JOSM. A new way was added with an inappropriate *lanes:forward* tag.

3. **Modify Simulation Settings**: The traffic planner can then use the control panel of TraLAMA (Figure 5.3) to define customized simulation settings such as the preferred simulation type, traffic demand or simulation length.

4. **Run Simulation**: When the traffic planner presses the *Save and Start* button, TraLAMA performs all necessary preprocessing steps for the simulation. Therefore, the defined simulation settings are required. Then TraLAMA automatically triggers a new SUMO simulation (Figure 5.4). If the simulation type is set to *graphical*, the *SimulationController* starts SUMO with a GUI parallel to JOSM. The traffic planner can then observe the simulation by following a specific vehicle, increasing or decreasing the simulation speed and modifying the coloring schemes for instance.

5. **Visualize Simulation Output**: When the simulation is finished, TraLAMA gets notified and the traffic planner can review the visual simulation results that is displayed as an additional layer in JOSM. This layer represents the GPX tracks of all vehicles and can be filtered by the traffic planner to only show a limited time frame. The simulation result is further parsed and sent to the TraLAMA web service, where the traffic planner can see another version of the simulation result, the traffic load map (Figure 5.5). In addition, the traffic planner

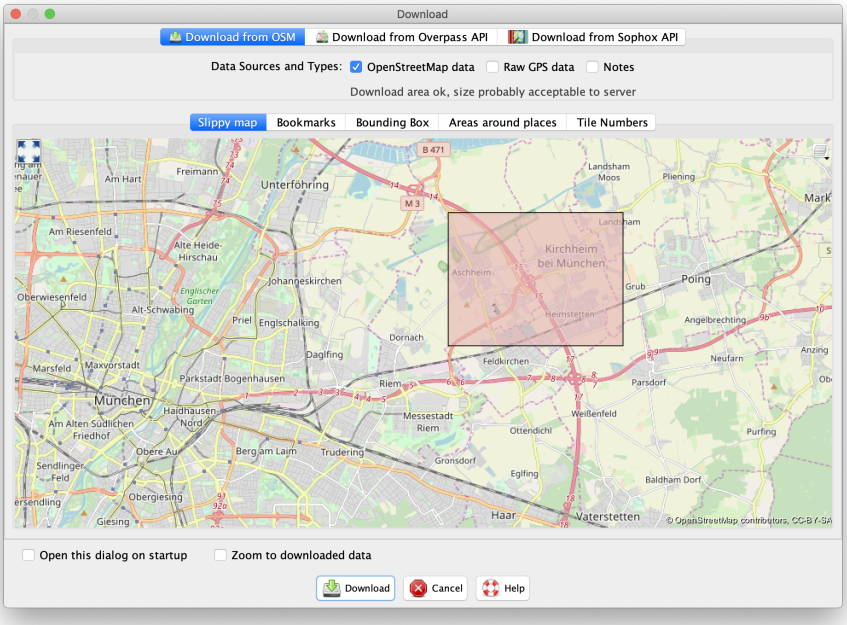


Figure 5.6: Download Map Data From OSM Web Server

has several options to customize the styling of this map (see Section 5.6).

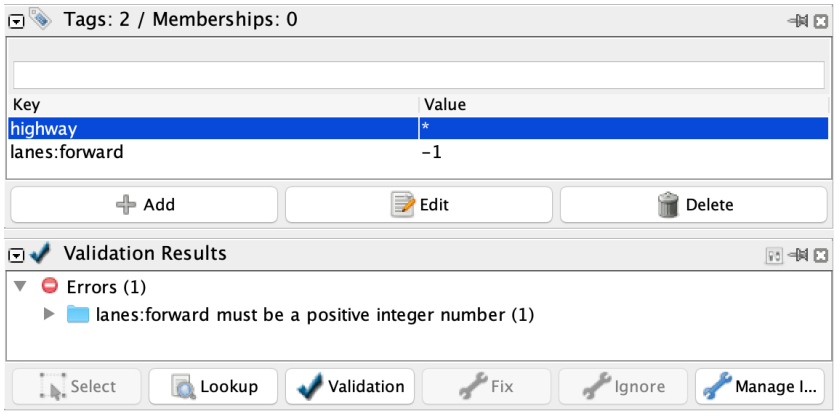


Figure 5.7: Validation Result of an Inappropriately Tagged OSM Way

Chapter 6

Evaluation

Initially, testing objectives are defined in Section 6.1. The subsequent evaluation methodology in Section 6.2 presents the testing environment and the testing procedure. In Section 6.3, three simulations scenarios are introduced with their results described in Section 6.4. The interpretation of these results, obtained findings and proposed solutions are then presented in Section 6.5. Lastly, arisen problems throughout the evaluation are discussed in Section 6.6.

6.1 Objectives

The aim of this evaluation is to find problems regarding the usability of TraLAMA. The following usability sub-criteria are derived from [JCB11].

- **Understandability**: Is the system easily understood?
- **Documentation**: Is the user manual comprehensible?
- **Learnability**: Is the learning curve appropriately?

To analyze these criteria, the subsystems (4.2) can be analyzed separately. The OSM editor JOSM, the developed JOSM plugin TraLAMA, the traffic simulation SUMO and the TraLAMA web service are evaluated.

6.2 Methodology

Three municipal employees of Kirchheim tested the implemented functionalities. These employees were responsible for the economic development and for the sectors 'mobility and projects' in the municipality Kirchheim.

The testing was performed in the construction and planning department of Kirchheim. The system was not deployed to their local workstations yet, but tested on a brought workstation that had been prepared to avoid potentially time-consuming setting up of environments.

Features of ACPS were then introduced and explained to the testers. This was an interactive process, where the testers raised questions and proposed new testing scenarios. Together we modeled two requested scenarios (see Section 6.3) using the JOSM editor. Then one tester modeled the third scenario by himself.

6.3 Simulation Scenarios

Different simulation scenarios within *Kirchheim* and *Heimstetten* were modeled and tested. The traffic demand for all simulations was the *morning* data.

- **Scenario 1**: Original road network data from OSM, no modifications
- **Scenario 2**: Added a new bypass road from *Kirchheim* to *Heimstetten*
- **Scenario 3**: Added a new bicycle road in *Kirchheim*

6.4 Results

6.4.1 Scenario 1 - Original Road Network

After simulating the first scenario, one tester discovered a road that was used by the simulation even though it should be inaccessible to cars. The simulation result is illustrated in Figure 6.1, where the dashed line represents the falsely used road. The road was tagged as follows:

- $highway = track$
- $surface = dirt$
- $tracktype = grade3$

Another important finding by the testers was that the simulation did not consider street types (2.3.1) for the calculation of *shortest* routes. Thus, the *residential* highway *Dorfstraße* was preferred over the *tertiary* highway *Erdinger Straße* because it led to a shorter travel time. According to the road traffic regulations, *residential* highways should be accessible to local residents only. They are not appropriate for transit purposes.

Figure 6.1: Simulating Vehicles Along Inaccessible Roads

6.4.2 Scenario 2 - Bypass Road

The second simulation was based on an additional bypass road. The simulation result of this scenario is illustrated in Figure 6.2. During the modeling process, one tester raised the question about the junction type used as the default junction in the simulation when no OSM tag is defined. As an empirical analysis showed, the simulation interprets junctions without a tag as *right-over-left*. However, this behavior is not stated in the documentation of the traffic simulation.

The result of the tested scenario was that the traffic did flow as expected through the newly created bypass road, since the road represented a shortcut from *Kirchheim* to *Heimstetten*. In addition, the testers easily understood the creation tool for adding new OSM ways to the road network. Furthermore, they could utilize the highway and junction presets and tag the new bypass road appropriately. During the simulation of the scenario, the testers were able to analyze the altered traffic flow.

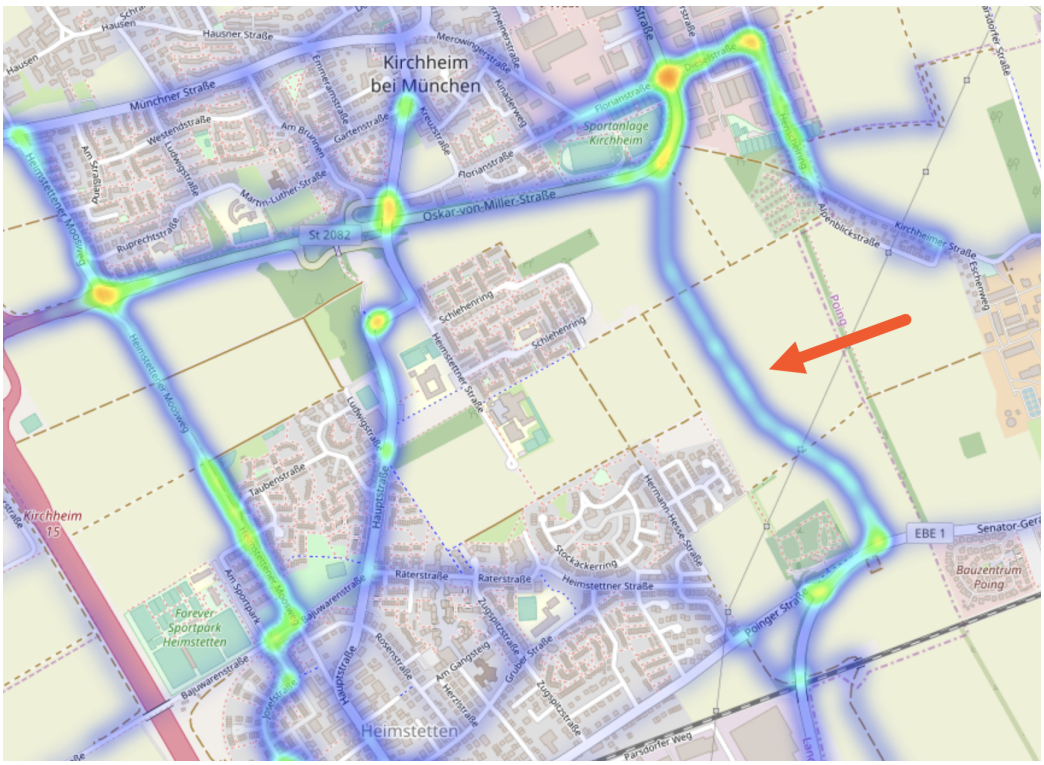


Figure 6.2: Bypass Road from *Kirchheim* to *Heimstetten*

6.4.3 Scenario 3 - Bicycle Road

For the third simulation, a tester modeled a bicycle road. As in scenario 1 (6.4.1), the simulation allowed cars to move along the newly created bicycle road, even though the road should be inaccessible to cars.

During the modeling process, the tester tried the different working modes in the *Edit Toolbar* of JOSM. He was able to add a new way, connect it to existing ones and assign bicycle tags to it. In addition, the tester customized the simulation settings by decreasing the simulation length and selecting random traffic demand. After he started the simulation, he reviewed the newly created bicycle road. As mentioned in the beginning, he discovered vehicles driving along this road.

6.4.4 Usability

Based on the input from the testers, the learning curve for TraLAMA appeared to be satisfactory. However, the testers also requested a user manual. The editing tools of the OSM editor to create new points or lines and assign tags to them worked properly. Only the zooming feature within the

editor was criticized as slightly difficult to use. One tester further requested multi-language support (German and English).

6.4.5 Traffic Load Map

Additionally, a tester asked for an improved integration of the TraLAMA web service functionality into the local application. The tester mentioned that a side-by-side comparison of the traffic load maps or a layered structure as in the OpenStreetMap editor would help them to better compare the simulation results.

6.4.6 Sensor Data

Besides, one tester presented an idea to improve the traffic demand data. Kirchheim could provide sensor data about the number of vehicles driving on four streets within the municipality.

6.4.7 Multimodal Simulations

Lastly, the testers wished the integration of multimodal traffic scenarios, in particular the inclusion of bicycle agents. Their goal is to find out the number of potential commuters who might swap their car for a bike when new cycle highways are introduced.

6.4.8 Summary

The overall feedback of the testers was positive. Two of them shared the opinion that TraLAMA has potential to analyze the immediate impact of road network editing on traffic flow. The third tester criticized the sense of purpose of TraLAMA for both city and traffic planners. The person argued that the modeling of new roads and junctions is too time consuming and that the municipality would need additional personnel to develop and assess novel transportation strategies using TraLAMA. On the contrary, one tester expressed deeper interest in TraLAMA and desired to have a portable version of it to play around with bypass roads during the evenings after work.

6.5 Findings and Discussion

6.5.1 Street Classification

To begin with, the results of the tested scenarios one and three (6.4) all refer to the route calculation of the vehicles to be simulated. The fact that routes were created which included roads that should be inaccessible to cars was shown to be an undesirable behavior. As well as the case where residential streets were preferred over tertiary ones. Therefore, together with the testers we propose the integration of already existing OSM tags [Ope19b, Ope19f, Ope19g] for highways such as *vehicle*, *surface* or *tracktype* into the route calculation. The first tag can be set to the value *no* and hence explicitly exclude vehicles. Using the second tag, an order can be defined that prioritizes surfaces such as *asphalt*, *concrete* or *paving stones* over *dirt*, *earth* and *grass*. The third tag represents dirt roads and forest tracks and can be specified as $grade\{i\}$ with $i \in \{1, 2, 3, 4, 5\}$, where the grading indicates its carrying capacity. One represents the highest and five the lowest capacity.

Besides, we suggest the integration of street types into the route calculation using the OSM *highway* tags, since it is equally important that main roads are preferred over *residential* or *living* streets.

6.5.2 Usability

The requested user manual was provided in the final stages of this thesis and multi-language support was implemented too. The delayed editing within the OSM editor can be partially attributed to the OSM editor itself and partially to the container platform Docker. However, no further time was spent in trying to improve the editing usability, since it was already a trade-off in Section 4.5.3.

6.5.3 Web Service, Sensors and Bicycles

The requested integration of the website was added to the product backlog but could not be implemented within the given time frame of this thesis. One has to mention that initially, the web service was not planned at all and was a way to overcome the limitations of trace styling options within JOSM. Therefore, the task was given low priority and later added to future work.

The vehicle count sensor data cannot be used for improved traffic demand at the moment because the traffic demand data used in TraLAMA is based on geographic *origin-destination* coordinates.

The integration of bicycle agents cannot be implemented either in a realistic way, since no statistical data about cyclists in Kirchheim is available at present.

6.6 Reliability

During the evaluation with the testers of Kirchheim, the web service did not work appropriately. TraLAMA sent the simulation result to the TraLAMA web service, but the web server could only extract the meta information of the simulation. As a result, the web view visualized an obsolete traffic load map with the meta information of the new simulation. Fortunately, the problem could be fixed directly on-site with a simple restart of TraLAMA.

Later it emerged that the JSON content body was too large for the server to handle. In order to enable simulation lengths of up to three hours, the implementation was updated to split large requests for the TraLAMA web service into multiple parts.

Chapter 7

Summary

This chapter reviews the final status (7.1) of this work. It presents the implemented functionalities as well as open goals for further development. The subsequent Section 7.2 describes limitations and their impact on the implementation and evaluation. Section 7.3 provides an outlook into possible future work. Lastly, Section 7.4 draws a conclusion and summarizes the contribution of this thesis.

7.1 Status

The system supports the import, modification and export of OpenStreetMap data. Moreover, static traffic demand could be generated and stored in such a way that traffic planners are able to select among different traffic scenarios. Additionally, the system integrates a traffic simulation and traffic planners can customize the simulation settings and review the simulation output in a visual illustration. Details about the implemented and open requirements are covered in the following.

7.1.1 Realized Goals

Table 7.1 shows the current status of the functional requirements and Table 7.2 illustrates the status of the nonfunctional requirements. Realized requirements are marked with ✓, partially implemented requirements with **!** and requirements that were not implemented yet and therefore represent open goals are indicated with **X**.

Name	Description	Status
[**FR1**] - Load Road Network	The system must provide a way to download and import OpenStreetMap data from the OpenStreetMap server.	✓
[**FR2**] - Modify Road Network	There must be an option to modify OpenStreetMap data, e.g. add new lanes, add traffic lights and traffic signs.	✓
[**FR3**] - Export Road Network	There must be a way to convert and export the modified OpenStreetMap data to the traffic simulation platform.	✓
[**FR4**] - Select Traffic Demand	Traffic planners must have the option to select from temporal traffic demand data.	✓
[**FR5**] - Customize Simulation Settings	The system should implement a graphical user interface to allow traffic planners to modify the simulation settings.	✓
[**FR6**] - Run Traffic Simulation	There must be a way to run a traffic simulation.	✓
[**FR7**] - Visualize Traffic Load Maps	Temporal traffic load maps must be generated and visualized using the traffic simulation output.	✓
[**FR8**] - Export Traffic Load Maps	There must be an option to export the generated traffic load maps.	✓
[**FR9**] - Show Simulation Statistics	Detailed simulation statistics must be created and visualized based on the traffic simulation output.	X

Table 7.1: Status of Functional Requirements

Name	Description	Status
[**NFR1**] Accuracy	The road network should be up-to-date and contain proper junctions.	✓
[**NFR2**] Usability	The entry point to the system should be enabled with a maximum of two clicks.	!
[**NFR3**] Usability	The graphical user interface and the traffic load map should be colorblind safe.	✓
[**NFR4**] Usability	The interaction between the OpenStreetMap editor and the traffic simulation should be enabled with one click.	✓
[**NFR5**] Portability	The system should be platform independent and able to run on Windows and macOS.	✓
[**NFR6**] Supportability	The communication point between the OSM editor and the traffic simulation should be based on the REST paradigm.	✓
[**NFR7**] Response Time	The creation process of the visualization of the simulation output must not take more than five minutes.	!
[**NFR8**] Packaging (Constraint)	The system should be packaged in a lightweight container.	✓
[**NFR9**] Legal (Constraint)	The system must be free and comply to the open-source standard.	✓
[**NFR10**] Interface (Constraint)	The system must support the usage of OpenStreetMap data.	✓

Table 7.2: Status of Nonfunctional Requirements

7.1.2 Open Goals

The following requirements are either not implemented or partially implemented at present. The reasons for this are discussed now.

FR9 **Show Simulation Statistics**: Detailed simulation statistics were given low priority throughout the course of this work because a subset of the statistics can already be reviewed during the running simulation in SUMO. As a result, the feature is still not implemented.

NFR2 **Usability**: Unexpected security issues occurred while trying to use a web application as the entry point for the system. The *Docker Compose*[1] tool is required to start ACPS and apparently, it cannot be executed with JavaScript due to issues in creating a child process.

NFR7 **Response Time**: The conversion process of parsing GPX traces into a simple CSV file can take several minutes, depending on the simulation length and the number of simulated vehicles. However, this process is necessary to visualize the traffic load map in the TraLAMA web view.

7.2 Threats to Validity

Several issues occurred throughout the implementation process of this work. Some of them could be bypassed or evaluated as additional trade-offs, whereas other limitations still affect the system at present.

- **Simulation Correctness**: The evaluation (Chapter 6) shows that although OSM mostly provides an accurate and appropriately classified road network, the transition to the traffic simulation has to be revised and improved. Otherwise, simulation agents such as vehicles, cyclists or pedestrians are routed along roads that should be inaccessible to them. This could lead to many cases that deviate from the reality.

- **Website as Starting Point**: As described in the previous section (7.1.2), security issues occurred while trying to execute a shell command from the web service, such as *docker-compose up* to start the system. Due to this fact, the decision was made to implement a different entry point for the system. Unfortunately, the adapted entry point differs from macOS to Windows.

[1] `https://docs.docker.com/compose/`

- **JOSM Heat Map Styling**: The options to customize the styling of GPX layers in JOSM were limited at implementation time and did not allow sufficient modification so that the results would have met the defined requirements. As a result, an additional web service was created that displays the traffic simulation output as a clearly recognizable traffic load map. The web service also features more styling options so that traffic planners can derive more valuable information of it.

- **Non Reachable Edges**: Another issue that occurred throughout the course of this work describes simulation routes which are discarded by the traffic simulation. This should not be the case since DUAROUTER was set to only create validated trips which are then used by the *SimulationRoute* tool. It is suspected that small deviations within the conversion processes between static and simulation routes (see Section 5.5.2) cause these *non reachable edges*, as they are defined by the traffic simulation.

- **Maximum File Upload Size**: In Section 6.6, it could be discovered that simulations with a length of more than 30 minutes and around 1000 vehicle agents usually cause simulation outputs in a size (at least 30 Megabytes) that is too large for an average HTTP POST request. To address this issue, the implementation was updated to split the JSON content body into multiple parts and send one HTTP request for each part.

7.3 Future Work

There are a number of features that could be still implemented. The following enumeration shows the most relevant ideas ordered by priority.

1. **Create and Reuse Persistent Traffic Demand Data**: The first goal is to let traffic planners create their own traffic demand. In doing so, static traffic demand could be generated to compare the impact of road network editing on traffic flows all over the world, not only in Kirchheim. To refine the traffic demand, population descriptions[2] could be further added to the generation process.

2. **Implement Multimodal Traffic Demand**: The second goal describes traffic simulations that include different road users as well, such as pedestrians, cyclists and public transport. In order to realize these

[2] `https://sumo.dlr.de/wiki/Demand/Activity-based_Demand_Generation`

multimodal traffic scenarios, the afore mentioned creation of persistent traffic demand has to be extended.

3. **Further Integration with SmartHeim²**: Simultaneously to this thesis, the *SmartHeim²* team was working on real-time traffic demand data for Kirchheim. Therefore, four vehicle count sensors were installed in the municipality to measure traffic flow. These sensors are placed along the main roads that cross the Kirchheim. Consequently, another great feature would be the integration of this real-time data into the developed system.

7.4 Conclusion

The *Agile City Planning Suite* enables traffic planners to easily edit and simulate traffic scenarios. By using OpenStreetMap data, which offers an intuitive way to modify the road network, and by providing visual feedback in the form of traffic load maps, the suite makes transportation planning more comprehensible.

Based on the static traffic demand for Kirchheim, traffic planners of the municipality can now analyze the impact of extended roads, bypass roads or adapted road junctions on traffic flow. As a result, the *Agile City Planning Suite* can be seen as a small step towards a congestion-free Kirchheim.

Besides, it seems reasonable that the TraLAMA plugin has a positive side effect on the OpenStreetMap platform. By encouraging traffic planners to directly work on OSM data instead of traffic simulation-specific road networks, incorrect and invalid OSM entries be found more easily and hence, complemented. This means that OpenStreetMap gets even more accurate, remains up-to-date and we all can use it further on, whether for looking up directions or navigation purposes.

Appendix A

Software Availability

The source code for the developed system can be downloaded using
git clone `https://repobruegge.in.tum.de/scm/dsm/jakobs.git`

- **Compatibility**: macOS Mojave and Microsoft Windows 10
- **Licenses**:
  - josm-img: GPLv3
  - sumo-img: EPLv2

A.1 Prerequisites

Docker[1] - Enterprise Container Platform for High-Velocity Innovation

A.1.1 Windows

- **XLaunch**[2] - X Server
- **Lifeboat**[3] - Lifeboat
- **Prepare XLaunch on Windows**
  Start XLaunch and select *Multiple windows*, *Display number -1*, *Start no client* and tick **all** *Extra Settings*. Then save the configuration file. This has to be done only once.

[1] `https://www.docker.com`
[2] `https://sourceforge.net/projects/vcxsrv/`
[3] `https://electronjs.org/apps/lifeboat`

A.1.2 macOS

- **XQuartz**[4] - X11
- **TraLAMA.app** - Can be found within the repository

A.2 Installing

1. Start Docker
2. Inside the *josm-docker* folder, build the JOSM image with **docker build . -t josm-img**
3. Inside the *sumo-docker* folder, build the SUMO image with **docker build . -t sumo-img**

The building of the SUMO image might take several minutes.

A.3 Starting

A.3.1 Windows

1. Start Docker
2. Remove the commands `/tmp/.X11-unix:/tmp/.X11-unix:rw` within the *docker-compose.yml* file. This has to be done only once.
3. Change the IP-address within the *docker-compose.yml* file to the IP address of your workstation.
4. Open Lifeboat, select the *docker-compose.yml* file and click the *Play* button on the upper right.

A.3.2 macOS

1. **Prepare XQuartz on macOS**
   When starting ACPS for the first time, go to the XQuartz settings and make sure to activate the *Allow connections from network clients* option. Then restart ACPS. This has to be done only once.

[4] `https://www.xquartz.org`

2. Double-Click the TraLAMA.app to automatically start Docker, XQuartz and ACPS. *Attention: Docker usually needs some time to enter the running state. Just open TraLAMA.app again.*

Note: The application requires the xhost executable within [5] and the docker-compose executable within [6].

A.4 Additional Information

DO NOT upload changes that do not represent the real world to the OpenStreetMap server. For example if you add a new traffic light for simulation purposes, an upload would cause falsified information for OpenStreetMap users unless the traffic authority places a real traffic light at this position.

A.4.1 Ports

JOSM Docker: Port 3030
SUMO Docker: Port 4040

A.4.2 Data

Network Data

OpenStreetMap data of Kirchheim can be found in `./data/`.

Demand Data

Static demand data for Kirchheim can be found in `./data/input-simulation/demand/`.

Output Data

Simulation output data can be found in `./data/output-simulation/`.

A.5 References

The implemented Docker files and the TraLAMA plugin are partially based on the following projects.

[5] `/opt/X11/bin/`
[6] `/Applications/Docker.app/Contents/Resources/bin/`

- **docker-sumo**[7] - Bo Gao
- **docker-josm**[8] - Young Hahn
- **sumoconvert**[9] - Ignacio Palermo, Julio Rivera

Data prepared by the SmartHeim[2] team is always marked with an **i**-prefix.

[7] https://github.com/bogaotory/docker-sumo
[8] https://github.com/mapbox/docker-josm-linux
[9] https://github.com/openstreetmap/josm-plugins/tree/master/sumoconvert

List of Figures

2.1 Traffic Simulation Strategies: Macroscopic, Mesoscopic, Microscopic 7
2.2 Road Network of Kirchheim 7
2.3 *WGS84* Ellipsoid Showing the Difference in Elevation 13

3.1 UML Use Case Diagram 20

4.1 UML Analysis Object Model 26
4.2 UML Subsystem Decomposition 28
4.3 UML Activity Diagram of the System's Workflow 29
4.4 Proposed Control Panel 30

5.1 TraLAMA Object Model 39
5.2 Modified OSM Road Network in JOSM 42
5.3 TraLAMA Control Panel 43
5.4 SUMO Simulation Based on a Modified OSM Road Network . 48
5.5 Traffic Load Map in the Web View 54
5.6 Download Map Data From OSM Web Server 56
5.7 Validation Result of an Inappropriately Tagged OSM Way . . 56

6.1 Simulating Vehicles Along Inaccessible Roads 59
6.2 Bypass Road from *Kirchheim* to *Heimstetten* 60

List of Tables

2.1 Street Type Classification in OpenStreetMap 11

3.1 As-is Scenario 1: Modify the Road Network 21
3.2 As-is Scenario 2: Run a Traffic Simulation 22
3.3 Visionary Scenario 1: Visualize Traffic Simulation Output . . 23

4.1 Review of Traffic Simulation Software 32

5.1 Exemplary CSV Entry for a Static Traffic Demand Trip 51

7.1 Status of Functional Requirements 66
7.2 Status of Nonfunctional Requirements 67

Bibliography

[AEE15a] Andrés F. Acosta, Jairo Espinosa, and Jorge E. Espinosa. Developing Tools for Building Simulation Scenarios for SUMO Based on the SCRUM Methodology. *Proceedings of the 3rd SUMO User Conference*, (May):23–35, 2015.

[AEE15b] Andrés F. Acosta, Jorge E. Espinosa, and Jairo Espinosa. TraCI4Matlab: Enabling the Integration of the SUMO Road Traffic Simulator and Matlab® Through a Software Re-engineering Process. In *Modeling Mobility with Open Data*, pages 155–170. Springer, Cham, 2015.

[Bar10] Jaume Barceló. *Fundamentals of Traffic Simulation*, volume 145 of *International Series in Operations Research & Management Science*. Springer, New York, NY, 2010.

[BBEK11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. SUMO – Simulation of Urban MObility - An Overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, Berlin, 2011. ThinkMind.

[BD09] Bernd Bruegge and Allen H Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 2009.

[BKM+15] Laura Bieker, Daniel Krajzewicz, Antonio Pio Morra, Carlo Michelacci, and Fabio Cartolano. Traffic simulation for all: A real world traffic scenario from the city of Bologna. In *Modeling Mobility with Open Data*, pages 47–60. Springer, Cham, 2015.

[BW14] Michael Behrisch and Melanie Weber. *Modeling Mobility with Open Data: 2nd SUMO Conference 2014*. Springer, Berlin, Germany, 2014.

[BW15] Michael Behrisch and Melanie Weber. *Simulating Urban Traffic Scenarios: 3rd SUMO Conference 2015*. Springer, Berlin, 2015.

[Deu17] Deutsche Bahn AG. Faktenblatt: Die S-Bahn München und ihr Streckennetz, dec 2017.

[GD12] Daniel Marques Gomes de Morais and Luciano Antonio Digiampietri. A review about multimodal traffic simulation techniques. *The Revista de Sistemas de Informação da FSMA*, 10:2–9, 2012.

[Ger19] German Aerospace Center (DLR) and others. FAQ - SUMO Documentation, 2019.

[HNA16] Andreas Horni, Kai Nagel, and Kay W. Axhausen. Introducing MATSim. In *The Multi-Agent Transport Simulation MATSim*, pages 3–8. London: Ubiquity Press, 2016.

[HW08] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 7(4):12–18, oct 2008.

[JCB11] Mike Jackson, Steve Crouch, and Rob Baxter. Software Evaluation: Criteria-based Assessment. *Software Sustainability Institute*, pages 1–13, 2011.

[JCG14] Remi Jedwab, Luc Christiaensen, and Marina Gindelsky. Rural Push, Urban Pull and... Urban Push? New Historical Evidence from Developing Countries. *The George Washington University, Institute for International Economic Policy Working Papers, 4.*, (January):45, 2014.

[KBW+15] Daniel Krajzewicz, Michael Behrisch, Peter Wagner, Raphael Luz, and Mario Krumnow. Second generation of pollutant emission models for SUMO. In *Modeling Mobility with Open Data*, pages 203–221. Springer, Cham, 2015.

[KH09] Gligor Kotusevski and Ken A. Hawick. A Review of Traffic Simulation Software. *Research Letters in the Information and Mathematical Sciences*, 13:35–54, 2009.

[KHRW02] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. SUMO (Simulation of Urban MObility) - an Open-Source Traffic Simulation. *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[KP15] Karl Heinz Kastner and Petru Pau. TOMS—traffic online monitoring system for ITS Austria west. In *Modeling Mobility with Open Data*, pages 189–201. Springer, Cham, 2015.

[Lan19a] Landesamt für Geoinformation und Landentwicklung Baden-Württemberg. GNSS, 2019.

[Lan19b] Landesamt für Geoinformation und Landentwicklung Baden-Württemberg. Koordinatenref.system, 2019.

[Ope19a] OpenStreetMap Foundation. Copyright and License, 2019.

[Ope19b] OpenStreetMap Wiki contributors. Attributierung von Straßen in Deutschland, 2019.

[Ope19c] OpenStreetMap Wiki contributors. Comparison of editors, 2019.

[Ope19d] OpenStreetMap Wiki contributors. Key:highway, 2019.

[Ope19e] OpenStreetMap Wiki contributors. Key:junction, 2019.

[Ope19f] OpenStreetMap Wiki contributors. Key:surface, 2019.

[Ope19g] OpenStreetMap Wiki contributors. Key:tracktype, 2019.

[PMS13] A. Pell, A. Meingast, and O. Schauer. Comparison Study of Software Tools for Online Traffic Simulation Supporting Real-time Traffic Management of Road Networks. In *Proceedings of 20th ITS World Congress*, number January, Tokyo, Japan, 2013.

[Pot12] Tomas Potuzak. Distributed-Parallel Road Traffic Simulator for Clusters of Multi-core Computers. In *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pages 195–201, Dublin, oct 2012. IEEE.

[RSR15] David Rieck, Björn Schünemann, and Ilja Radusch. Advanced traffic light information in openstreetmap for traffic simulations. In *Modeling Mobility with Open Data*, pages 25–34. Springer, Cham, 2015.

[Sta19] Statista GmbH. Anzahl der gemeldeten Pkw in Deutschland - PKW-Bestand bis 2019, 2019.

[The18] The World Bank. Urban population (% of total population), 2018.

[Tra14] Transportation Research Circular. Looking Back and Looking Ahead: Celebrating 50 Years of Traffic Flow Theory, A Workshop. *Traffic and Transport Simulation*, (April), 2014.

[Tur19] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2019.

[Uni18] United Nations. 68% of the world population projected to live in urban areas by 2050, 2018.

[Wor18] World Health Organization. Ambient (outdoor) air quality and health, 2018.

[Zil18] Michael Zilske. *Transparent and versatile traffic simulation: Supply data, demand data, and software architecture*. Doctoral thesis, Technische Universität Berlin, 2018.

[ZNKE17] Daniel Zehe, Suraj Nair, Alois Knoll, and David Eckhoff. Towards CityMoS: A Coupled City-Scale Mobility Simulation Framework. In *5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication (FG-IVC 2017)*, Erlangen, Germany, apr 2017. FAU Erlangen-Nuremberg.

[ZNN11] Michael Zilske, Andreas Neumann, and Kai Nagel. OpenStreetMap For Traffic Simulation. *M. Schmidt, G. Gartner (Eds.), Proceedings of the 1st European State of the Map – OpenStreetMap conference, no. 11-10*, pages 126–134, 2011.