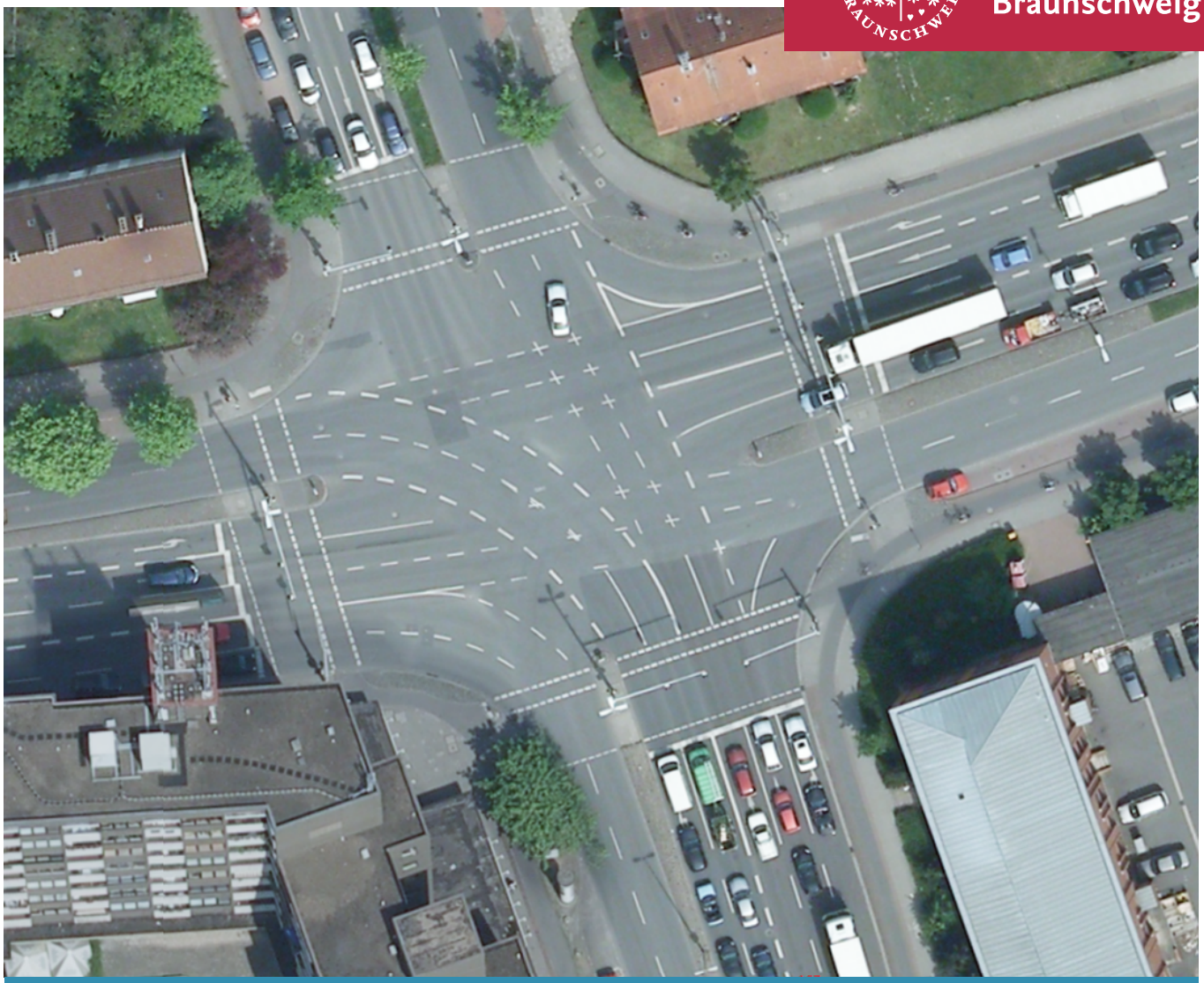
Chair for
Chip Design for
Embedded Computing

Technische
Universität
Braunschweig



Methods for integrating control algorithms of light signal systems for various simulation environments with special consideration of public transport

Danny René Behnecke

January 30, 2017

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged.

Date: Signature:

Acknowledgements

First I would like to thank Prof. Dr.-Ing. Mladen Berekovic for supervision of my thesis and enabling me to write it at the German Aerospace Center/Deutsches Zentrum für Luft- und Raumfahrt (DLR). Furthermore my thanks belong to Bastian Farkas who did a great job of advising me and supporting me while writing this thesis.

I also would like to thank Dr.-Ing Martin Fischer and Dirk Assmann of the DLR for supervising me directly while developing the programs for the thesis. Their input was very helpful and I couldn't have done this without their support.

I also would like to thank my dear colleagues at DLR for not only providing me with all the information needed for this thesis but also for their continuous support and encouragement.

Lastly I must express my gratitude to my parents and to my girlfriend Maria Baldauf for supporting me and caring for me throughout my years of study and through the writing phase of this thesis. Without being backed by them all of this couldn't have been accomplished. Thank you.

Danny Behnecke

Zusammenfassung

Simulation ist ein wesentlicher Bestandteil heutiger Forschung und Entwicklung aufgrund ihrer hohen Effizienz. Den größten Einfluss hat sie in Bereichen in denen ohne sie entweder hohe Verluste oder sogar Leben auf dem Spiel stünden. Durch manuelles Testen würde man auf lange Sicht beides unnötig verschwenden. Diese Faktoren sind besonders ausschlaggebend im Bereich der Verkehrsforschung, genauer, der Verkehrskontrolle. Ein Verkehrsleitsystem welches Fehlfunktionen aufweist könnte verschiedenste Folgen haben, die bei einem einfachen Stau beginnen und im schlimmsten Fall bis zum Verlust von Menschenleben reichen. Ein Stau wiederum hätte, sofern diese Fehlfunktionen andauern, Einfluss auf die lokale Wirtschaft und könnte ein treibender Faktor gegen einen Wirtschaftsstandort sein. Um all dies zu vermeiden ist es unabdinglich, dass moderne Verkehrsleitsysteme Konsistenz und Ausfallsicherheit aufweisen. Weiterhin werden die Methoden, mit denen Verkehr geleitet wird immer vernetzter und komplizierter. Das bedeutet, dass in diesen Qualitätssicherungsprozess auch die Hardware der betreffenden Infrastruktur mit eingebunden werden muss um nicht nur die Hardware selbst zu testen, sondern auch sicher zu stellen, dass sie potent genug ist das gewählte Verfahren überhaupt hinreichend schnell und sicher berechnen zu können. Diese Arbeit präsentiert ein Design für eine skalierbare und flexible Architektur einer Verkehrssignalsteuerung für das Simulationsframework DOS des DLR, eine Möglichkeit in dieses Programme mit SUMO erstellt zu importieren, sowie eine Analyse wie Hardwaretests in die vorhandene Forschungs- und Entwicklungspipeline eingebunden werden könnten. Weiterhin wird die präsentierte Architektur diskutiert um nicht nur zu zeigen, dass sie wirklich skalierbar und flexibel ist, sondern auch wie mit ihr Kontrollmethoden, welche den öffentlichen Nahverkehr besonders berücksichtigen, sehr direkt implementiert werden können. Zu guter Letzt werden Vorbereitungen präsentiert für ein zukünftiges Feature der Simulationsumgebung am DLR, ein Livestream der Ampel- und Verkehrsdaten der Braunschweiger Referenzstrecke in ihr virtuelles Gegenstück.

Schlagworte: Simulation, Dominion, SUMO, Hardware Test, Software Architektur, Design Pattern

Abstract

Simulation is a crucial part of today's research and development due to its high efficiency. It is mostly effective in environments where otherwise high costs or even lives were to be put on stake for testing new approaches. This is both the matter of fact when facing the domain of traffic control. If a traffic light system is malfunctioning the result can vary from a traffic jam, which would create an economic loss on a regional level. This might increase over time if this malfunction is persistent and even lead to accidents that cost lives. For example when all lights at an intersection show green the same time. To avoid all that it is very important to test everything that goes live on real hardware beforehand and ensure its reliability and fail-safety. Furthermore, the more complex the mechanics to control traffic lights and thus traffic flow in general become too. The existing hardware, especially the controllers, should be integrated into this testing pipeline to estimate the effects of hardware lifespan on the deployed algorithms and to proof that it can actually run the algorithms. This thesis provides a prototype and a design for implementing a scalable and flexible architecture for traffic light signal systems in the DLR's simulation framework DOS, a possibility to import traffic light programs created with the simulation framework SUMO and examining how hardware testing could be a future feature of DLR's test environment. Furthermore it's discussing how the presented architecture is not only living up to the promise of scalability and flexibility, but also how control schemes with public transportation consideration can be straight forward deployed on it. Lastly, it presents a preparation for a future feature of the simulation environment, the live-stream of traffic light data from the city of Braunschweig to its virtual counterpart.

Keywords: Simulation, Dominion, SUMO, Hardware Test, Software Architecture, Design Pattern

Contents

1. Introduction **1**
1.1. Goal . . . 1
1.1.1. Research problems . . . 2
1.2. Structure of the Thesis . . . 2

2. Background **3**
2.1. Light signal systems . . . 3
2.2. Controlling Approaches . . . 4

3. Environment **5**
3.1. Hardware Environment . . . 5
3.1.1. AIM - Application Platform for Intelligent Mobility . . . 6
3.1.2. VR-Lab - Virtual Reality Laboratory . . . 6
3.1.3. MoSAIC - Modular and Scalable Application-Platform for Testing and Evaluating of ITS Component . . . 6
3.1.4. Reference Track and Research Intersection . . . 7
3.2. Software Environment . . . 7
3.2.1. SUMO - Simulation of Urban MObility . . . 7
3.2.2. Dominion . . . 9
3.2.3. VTD - Virtual Test Drive® . . . 10

4. State of the Art **13**
4.1. Current Practice . . . 13
4.1.1. SUMO . . . 13
4.1.2. DOS . . . 13
4.1.3. VTD . . . 14
4.2. Traffic Signal Control . . . 14

5. Design **17**
5.1. Requirements and long-term Design . . . 17
5.2. Fixed-Time Programs . . . 18
5.2.1. Retrieving programs . . . 19
5.2.2. **Conversion Metric** . . . 19
5.2.3. Running programs . . . 21
5.2.4. Connection to Dominion . . . 22
5.2.5. Fixed time control strategies . . . 22
5.3. Adaptive Programs . . . 22
5.3.1. Retrieving Programs . . . 22
5.3.2. Running programs . . . 23
5.3.3. Adaptive time control strategies . . . 23

5.3.4. Special Focus: Public Transport 24
5.4. Interfacing the Reference Track and Research Intersection 24

6. Implementation **27**
6.1. Prerequesits and Disclaimer 27
6.2. TLConfigLoader 27
6.2.1. LightControl 29
6.3. SPaTInterface 30
6.4. TLGrabber 32
6.4.1. Requirements 32

7. Evaluation **35**
7.1. Meeting the software requirements 35
7.2. Evaluation of hardware requirements 37
7.2.1. Simulating Hardware 38

8. Conclusion and Outlook **39**
8.1. Outlook 39
8.2. Code Improvements 39
8.2.1. Functional Improvements 39
8.2.2. Design Pattern Improvements 40
8.2.3. Refactoring 40

Appendices **49**

A. Observer Design Pattern **51**

B. Model-View-Controller Pattern **53**

C. Design Overview **55**

D. Class Diagrams **57**

E. Short instructions **65**
E.0.1. Prepare map for SUMO 65

1 Introduction

With ongoing urbanization of states and more and more people moving to cities thus intensifying conurbations, traffic will become a serious issue in the future. There still have to be found solutions to a myriad of problems regarding flow-control, composition of traffic in specific areas and how it can be applied with a focus on impact on humans [12]. Therefore, simulations have to be done to not only prevent people getting harmed by experimental processes but also to validate the human compatibility. Furthermore there are a myriad of ways to control traffic lights, each and every one with it owns pros and contras, which have to be evaluated to find an optimal solution for the needed situation. To do these simulations requires a test infrastructure that gets as close as possible to reality.
The Institute of Transportation Systems of the DLR is constantly working on improving their infrastructure to achieve these circumstances. When dealing with traffic flow control one of the ways that need most attention within big cities is the control of traffic lights. Traffic lights are arguably the signals that have the biggest impact on the flow of inner city traffic since they are responsible for preventing crashes at junctions and controlling the flow of traffic.
Up until now the DLR's environment lacks a generic interface to interact with their simulated traffic lights in a way that doesn't need several people to setup a scenario, which might take at least a week, before being able to define just a simple traffic light program. Since traffic lights can have a huge impact on drivers and their behavior, their programs have to be adjusted with an ease that tunes down the complicity of the simulation to go further up in abstraction levels of traffic control. To achieve this a general interface will be introduced to not only implement fixed-time controls but also create an interface for complex, adaptive control mechanisms.
Besides the simulation aspect of traffic research it has also to be said that the DLR and the city of Braunschweig have a unique agreement that the DLR is allowed to monitor and collect traffic related data, varying from traffic light states to images and videos of real traffic, within the so called Reference Track (see subsubsection 3.1.4 at page 7). This enables the researchers at DLR to not only do simulations but also to do analytical research of the traffic in Braunschweig. For the future there is a feature planned to connect these two research domains with mirroring the real traffic light states into a virtual model of Braunschweig to not only have an overview over all states on a planning perspective, but also to analyze the correlation between traffic light program and traffic flow more thoroughly.

1.1. Goal

The goals of this thesis are to provide an interface for importing traffic light control programs for the DOMINION framework from Simulation of Urban MObility (SUMO). In addition to that designs will be introduced for implementing adaptive control strategies for traffic lights in a simulation environment. Furthermore, a tool for logging and observing the states of the traffic lights used by the Application Platform for Intelligent Mobility (AIM) (see 3.1.1 at page 6) will be introduced.

1.1.1. Research problems

These are the problems that were given for this thesis:

1. Analysis of mapping of traffic light control in Dynamic Object Simulation (DOS), Virtual Test Drive® (VTD®) and SUMO
2. Design and implementation of an application that interfaces to all respective software modules based on netconvert[1]
3. Enhancement of netconvert[1] to import recorded traffic light data and adaptive control schemes
4. Analysis of mapping of traffic light controls for high complexity intersections
5. Analysis of live-stream from real infrastructure
6. Analysis of hardware test for created algorithms

1.2. Structure of the Thesis

First, in chapter 2 is the background described about traffic light systems and how they can be controlled. After that, in chapter 3 will be shown for what environment the research problems are evaluated. In chapter 4 is described what the state of the art was before this thesis was started. It shows why a generic transformation is useful for the tools used at DLR. It also shows that with a continuously adaption of this thesis' design the threshold for designing and altering traffic scenarios in DOS will lower significantly, due to the generalization of the process that than will not need in-depth knowledge of the technology anymore.
The chapters 5 and 6 show what architecture is used for the general traffic light control and how the interfaces are composed to feature interoperability between the different software frameworks. This includes both the transformation of traffic light programs for DOS as well as future usage of recorded phase information from real infrastructure. Furthermore, in section 5.2.2 in chapter 5 can be found a discussion how to use the presented design for complex intersections.
In chapter 7 can be seen what extra requirements where given. Also it discusses the possibility of hardware tests and live-stream from real infrastructure. Lastly, chapter 8 provides a conclusion and the outlook on what should be done in succession to this thesis.

[1] The naming of netconvert was a communication mistake at formulating the thesis problem statement, which was corrected in a verbal agreement.

2 Background

This chapter introduces background information to illustrate the environment of the thesis and the domain in which the problems it solves lies.

2.1. Light signal systems

Light signal systems are a fundamental part of modern traffic control infrastructure and provide the highest priority of guidance in traffic when no police officer is in place.Traffic lights have been initially invented by John Peake Knight [2] and implemented in 1868 but were quickly neglected after an accident with the then used gas lights [1]. About 50 years later they got 're-invented' in the USA where the first permanent installation took place in 1914. And less than a decade later the traffic light began to spread to Europe [10]. Traffic lights have two major functions: regulating the flow of traffic within the road network in a city and providing a mechanism to establish safety on intersections. For every intersection exists a program or phase plan that dictates which signal has which state at what time. Those programs are either hand-made or designed with software tools. Either way those designs have to fit the rules so no two competing lanes are cleared for driving at the same time. When speaking about intersections and traffic lights some vocabulary has to be defined.

Intersection A crossing of two or more roads. An example for a standard four way intersection is shown in 2.1.

Lane A road can have multiple lanes for different directions to turn to or just to hold more traffic at once.

Path The direction in which a car is legally allowed to turn at a junction.

Traffic Light A traffic light is the physical entity that shows the phase through colored light. They can be seen by the colored lines in 2.1.

Signal Group A signal group combines all traffic lights that share the same phase at the same time. In 2.1 typical signal groups are shown: For each arm of the intersection there are basically two cases to be taken into account. Case 1 (as shown) for those vehicles driving straight or turning right and case 2 for turning left. Since opposite arms do not interfere with each other when vehicles are driving straight or right they can be grouped together as well as left turners which ultimately sums up to 4 signal groups on a standard junction: green straight + right, green left, red straight + right and red left.

Signal System The combination of all signal groups at one intersection.

Phase The meaning behind a color displayed by a traffic light. The common phases for cars are stop (=red), attention (=yellow), stop-attention (=red-yellow) and go (=green).

Program The sequence in which phases will take turns.

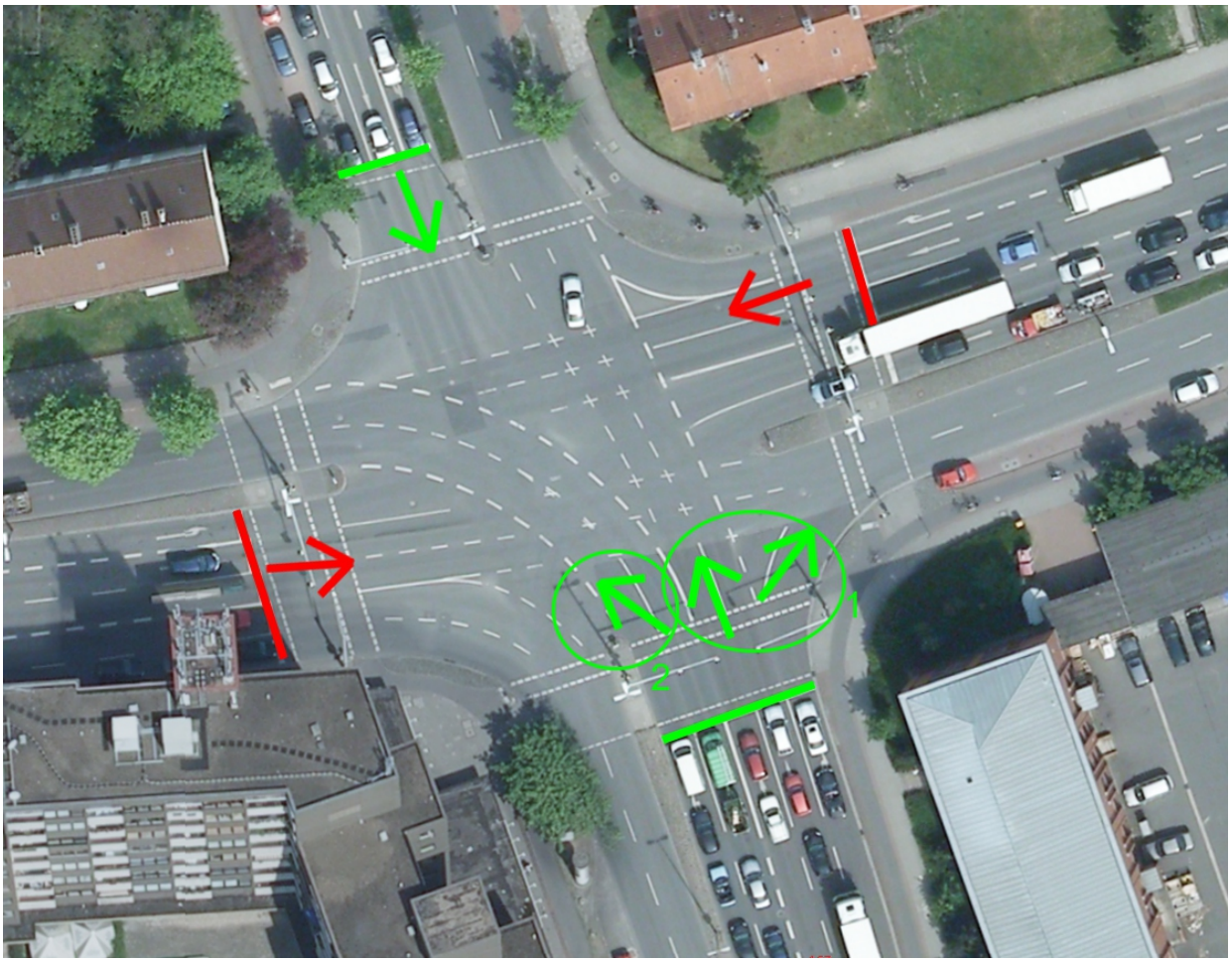


Figure 2.1.: Research Intersection in Braunschweig (Source DLR (CC-BY 3.0))

2.2. Controlling Approaches

Traffic light controlling has evolved over many years and now there are quite a few strategies available to tackle this problem. Basically this can be divided into three categories: fixed-time, adaptive-time approaches and a combination of both. While fixed-time approaches are very simple to implement and to maintain they lack the flexibility a typical urban environment needs. There are some ways to make up for that initial stiffness though, for example using several different fixed-time programs to adjust to statistical detected variances according to the day of the week or the current daytime. This would be for example to prioritize the incoming traffic in the morning and the outgoing in the evening. Although this helps already for smaller sized cities it does not solve all problems in bigger ones, especially not in traffic hubs that are located at junction points of major roads. Therefor the adaptive-time approaches emerged. They are a more reactive than planned manner to deal with traffic. Most of the time they have some thresholds for minimum and maximum phase length and when to change phase, although there are already methods that modify that too [9]. Last but not least there are combinations of the former depicted, where some adaptiveness is embedded into a greater plan that works network wide [11] [8].

3 Environment

Every domain of research has its own environment in which it operates and for traffic research beside the real world it is simulators, either software or hardware implemented. Simulation in general is done to increase the number of tests by simultaneously decreasing the cost per test. A high number of tests is needed to obey the law of large numbers of probability theory - the higher the number of test cases is, the higher the probability to approach the real values. This is crucial to research with humans since the basis of psychological studies are statistics. Another reason is, as sketched in the motivation, the minimization of potential harm to probands and environment. To achieve this the DLR has built several smaller and larger simulators in which humans can be immersed into the simulation scenario on several different levels.
Two of these simulators will be presented in the following sections. Also the infrastructure of Application Platform for Intelligent Mobility (AIM) will be presented as an alternative source for traffic light data and for providing another focus in traffic research. But with hardware alone it's impossible to run simulations and research peoples behavior to traffic, you need software to achieve that. Three different software frameworks will be presented, used and enhanced for this thesis, namely SUMO and Dominion which are in-house developments of the DLR and VTD® by VIRES Simulationstechnologie GmbH. These frameworks are used in different laboratories at DLR for creating an immersive and as believable as possible environment to simulate a traffic scenario that is on par with the real traffic environment. Also the Reference Track and Research Intersection that's used to learn more about traffic will presented and with that knowledge to improve simulation of it.

3.1. Hardware Environment

This section presents the hardware environments used at DLR and introduces the umbrella project 3.1.

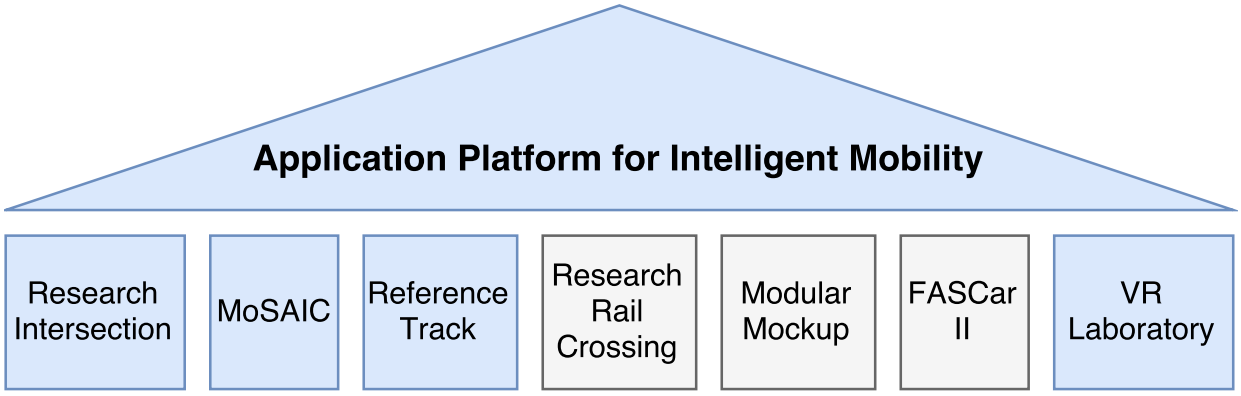


Figure 3.1.: Overview AIM

3.1.1. AIM - Application Platform for Intelligent Mobility

With AIM the DLR has taken a step towards accessibility of public-generated data for research by making the public itself a research object. AIM's major research priorities are traffic flow optimization, intermodal mobility, future mobility concepts, introduction of new and migration from existing systems, and mobility awareness. With these priorities in mind, the hereafter described laboratories were created to challenge the problems that arise from those areas. This thesis itself also fits into this spectrum, especially the introduction and migration field.

3.1.2. VR-Lab - Virtual Reality Laboratory

The VR-Lab is an environment where a virtual reality can be combined with the actual physical realization of a vehicle, like a car or a train. As sketched in the figure 3.2 its core piece is a cylindrical dome in which said vehicle can be placed and connected to the Back-End hardware of the system. While sitting in the vehicle surrounded by 360° projector images showing you the virtual world you're interacting with. Hardwarewise the laboratory

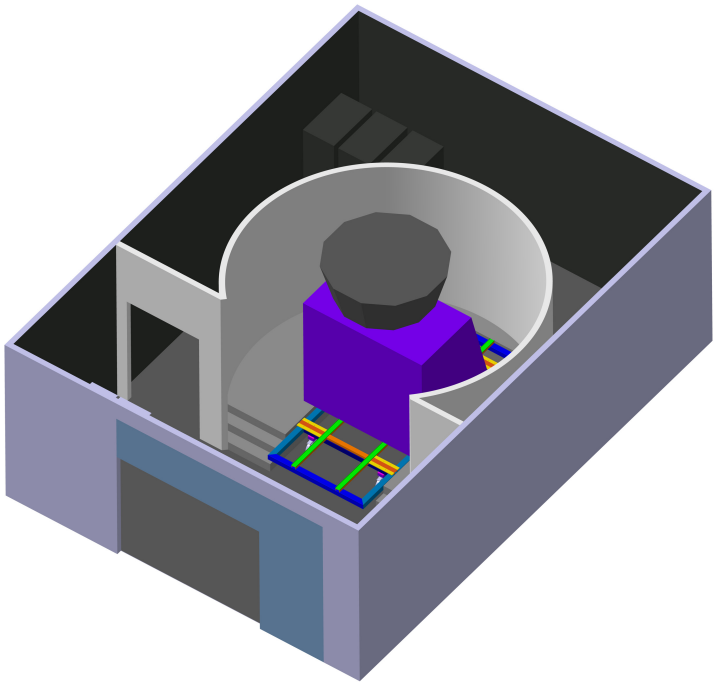


Figure 3.2.: VRLab

is equipped with 22 servers responsible for various tasks from projection over virtualization handling of the hardware to audio control and simulating the world. This enables researchers to conduct studies with one driver in a real-time simulated environment with full immersion for the test subject and so getting close to real-life circumstances.

3.1.3. MoSAIC - Modular and Scalable Application-Platform for Testing and Evaluating of ITS Component

The MoSAIC laboratory is a testing environment for multiple humans in the loop experiments. To achieve this it has three separate simulation setups linked up to another (see

figure 3.3). This enables the researcher not only to interact and study three different drivers at the same time but also to learn about the interactions amongst the drivers. As shown in 3.3 every setup has three monitors, a driving seat, steering wheel, pedals and a shift stick as its core. Although it takes away the immersive fact of sitting in a real car, the approximation is sufficient for a wide range of tests.

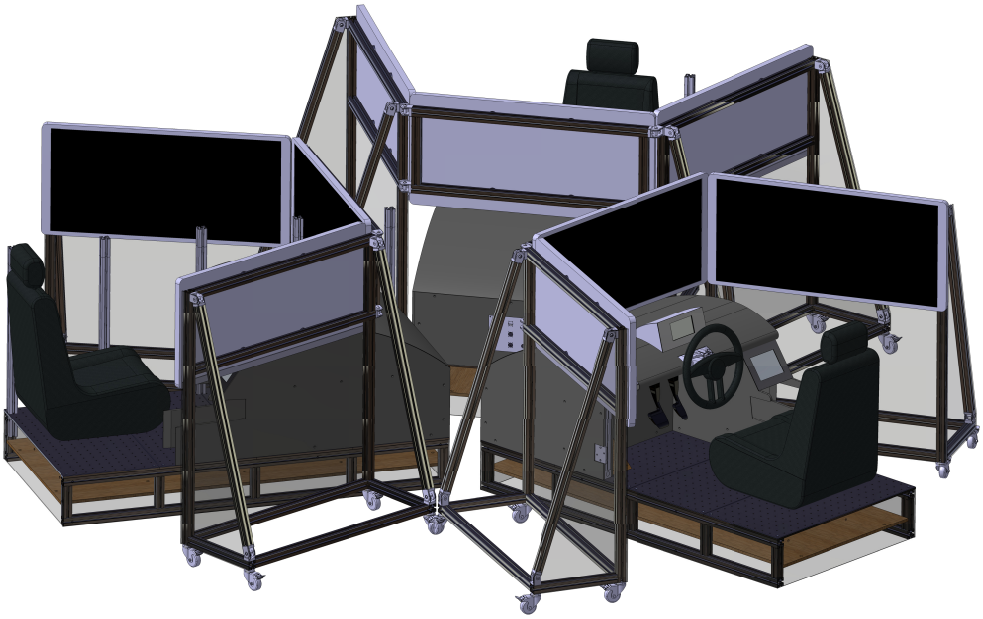


Figure 3.3.: MoSAIC Setup[5]

3.1.4. Reference Track and Research Intersection

The Reference Track refers to the "'City-Ring"' in Braunschweig which was equipped with so-called Road Side Units (RSU) that are the interface to all the data collected at each intersection. Every intersection can be queried for its traffic lights status. One intersection was equipped with more than that, the so-called Research Intersection has also cameras installed for traffic observation, which is located in the north-eastern corner of figure 3.4, indicated by a red circle. Both systems provide researchers an unique way of studying the traffic in the city of Braunschweig and at an exemplary intersection. This way both global and local effects will be able to be understood better in the future.

3.2. Software Environment

This section describes the used software frameworks which are involved with this thesis. Mainly it will be concerning the package DOS from Dominion, whereas VTD® will only play a minor role due to circumstances throughout the development cycle.

3.2.1. SUMO - Simulation of Urban MObility

According to [13],

SUMO is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. In fact, SUMO is a tool that allows the user to influence wide aspects of a traffic simulation, be it the traffic density or individual vehicle

Figure 3.4.: Main part of the Reference Track (green line) (Source DLR (CC-BY 3.0))

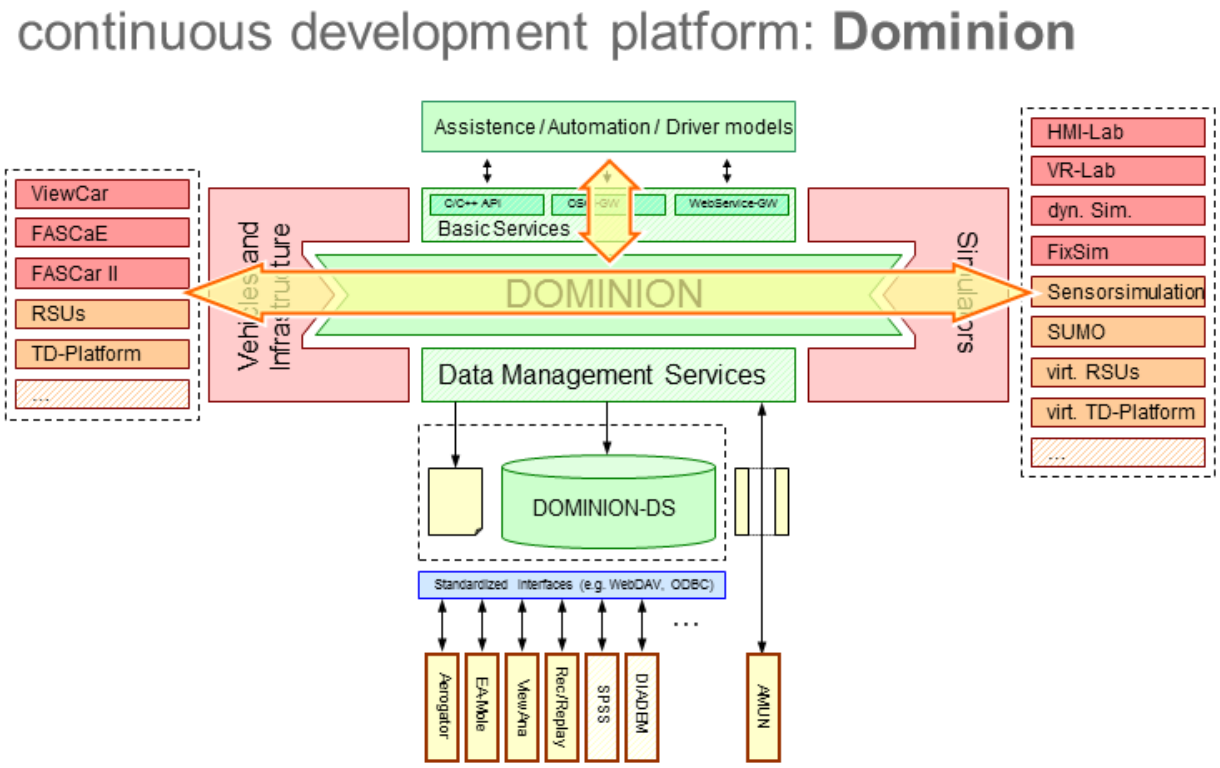


Figure 3.5.: Dominion Overview (Source DLR (CC-BY 3.0))

behavior. Also SUMO provides a quite sophisticated set of tools for planning the road network or import already specified maps. Upon that, it has the feature to automatically generate traffic light configurations which will be the starting point for this thesis.

3.2.2. Dominion

Dominion is the in-house solution of the DLR for simulating traffic scenarios in 3D virtual environments. It is a framework handling all inter-process communication of all registered programs which can be running decentralized over a network. A common use-case for the laboratories is that every big process has its own machine to run on to prevent deadlocks and lags while the simulation is running. An overview of DOMINION (Dominion) can be seen in figure 3.5. It works as a general framework and interface between the various used laboratories, vehicles and infrastructures. This framework handles all data management and communication between services and structures. Dominion is a model based framework which is able to generate its basic structures just by defining them in the so called data core. With this definitions it is able to create application models through a model-to-model-transformation and then source code via a model-to-text-transformation as depicted in figure 3.6. The communication between components within Dominion works through the Dominion Server which manages the data core. This server can run dedicated in the network or together with applications on the same machine. The Applications communicate with the server through UDP or if they are not running on the same machine. If they are though, they communicate via shared memory to reduce traffic and delays. The communication through UDP works with unicasts as well as with broadcasts, so that dif-

ferent multiplicities are realizable (see figure 3.7).

Dynamic Object Simulation (DOS)

The module DOS is responsible for simulating everything that can move around in the simulation. It is the in-house variant to the traffic simulation of VTD® and will be the main module this thesis' project will be tested with.

3.2.3. VTD - Virtual Test Drive®

> VIRES VTD® is a complete tool-chain for driving simulation applications: Starting from the definition of road networks with our road designer "ROD", we provide a consistent data flow via industry standard file formats into our modules for scenario definition, traffic simulation, image generation, sound simulation, data processing etc. VTD® provides open interfaces for 3rd party components and a plug-in concept with API for 3rd party modules. VTD® is in service in numerous applications in the automotive industry. [7, p. 1]

VTD® is furthermore able to be used modular so you just use the part of the tool-chain you are actually using. This will be the case in this thesis VTD® is only used as alternative viewer.

Figure 3.6.: Model based design of Dominion (Source DLR (CC-BY 3.0))

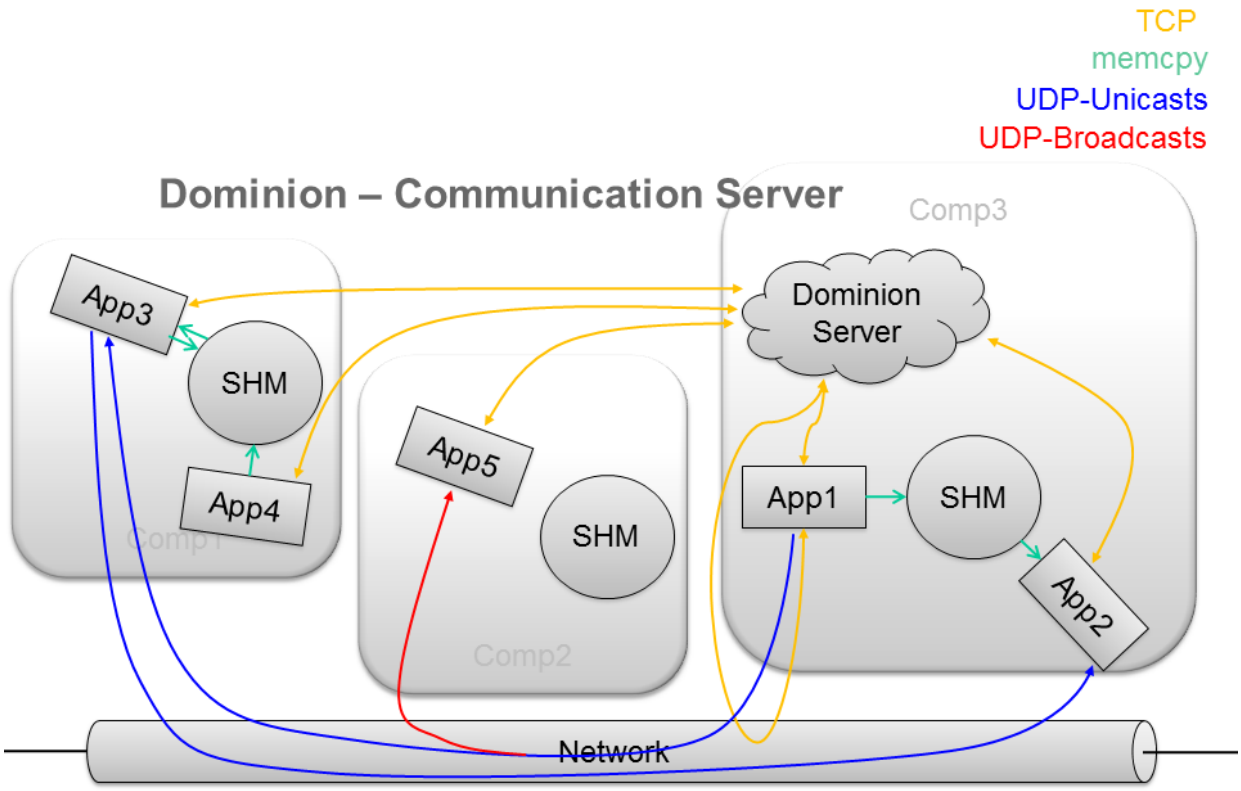


Figure 3.7.: Example of communication in Dominion (Source DLR (CC-BY 3.0))

4 State of the Art

This chapter describes the current state of the art for handling traffic lights in the simulation environment and why it is useful to implement more automated solutions to ease that process.

4.1. Current Practice

Currently most of the traffic light generation and setup has to be made manually, at least for Dominion and VTD. SUMO on the other hand has already means to generate random traffic lights for each intersection - this is the starting point for this thesis.

4.1.1. SUMO

SUMO is used for macroscopic traffic simulation with pre-determined scenarios being a special use-case. It is a standalone software with support for Vires' data formats and a connection possibility to VTD®. Up until now SUMO's usage was separated from Dominion, as traffic simulation and scenario simulation for human-in-the-loop (HIL) testing are two different domains to begin with. Traffic lights can be defined in SUMO through the program netedit which ultimately will write the definition of the programs with their respective phases into their own XML tags nested in the sumocfg.xml format which SUMO uses.

```
<tlLogic id="GS_10" type="static" programID="0" offset="0">
        <phase duration="31" state="rrrrGGGgrrrrGGGg"/>
        <phase duration="4" state="rrrryyygrrrryyyg"/>
        <phase duration="6" state="rrrrrrrGrrrrrrrG"/>
        <phase duration="4" state="rrrrrrryrrrrrrry"/>
        <phase duration="31" state="GGGgrrrrGGGgrrrr"/>
        <phase duration="4" state="yyygrrrryyygrrrr"/>
        <phase duration="6" state="rrrGrrrrrrrGrrrr"/>
        <phase duration="4" state="rrryrrrrrrryrrrr"/>
    </tlLogic>
```

Listing 4.1: Example of SUMO program definition

4.1.2. DOS

DOS has, as in-house solution for simulating traffic for scenarios, almost everything defined explicitly in one place, the scenario description. Although this has its justification with respect to the general design idea it is not the most usable concept to work with.

```
<SignalSystem idOnTrack="2" SPaT_intersectionId="2">
      <SignalGroup coreId="0" idOnTrack="1" SPaT_movementID="0">
        <Signals>
          <Signal idOnTrack="31" SPaT_laneId="31"/>
          <Signal idOnTrack="32" SPaT_laneId="32"/>
          <Signal idOnTrack="41" SPaT_laneId="41"/>
          <Signal idOnTrack="42" SPaT_laneId="42"/>
```

```
        </Signals>
        <Behaviour>
          <TimeOffset>0</TimeOffset>
          <PhaseDuration type="Green">10</PhaseDuration>
          <PhaseDuration type="Yellow">2</PhaseDuration>
          <PhaseDuration type="Red">14</PhaseDuration>
          <PhaseDuration type="RedYellow">2</PhaseDuration>
        </Behaviour>
      </SignalGroup>
      <SignalGroup coreId="1" idOnTrack="3" SPaT_movementID="3">
        <Signals>
          <Signal idOnTrack="35" SPaT_laneId="35"/>
          <Signal idOnTrack="36" SPaT_laneId="36"/>
          <Signal idOnTrack="45" SPaT_laneId="45"/>
          <Signal idOnTrack="46" SPaT_laneId="46"/>
        </Signals>
        <Behaviour>
          <TimeOffset>14</TimeOffset>
          <PhaseDuration type="Green">10</PhaseDuration>
          <PhaseDuration type="Yellow">2</PhaseDuration>
          <PhaseDuration type="Red">14</PhaseDuration>
          <PhaseDuration type="RedYellow">2</PhaseDuration>
        </Behaviour>
      </SignalGroup>
    </SignalSystem>
```

Listing 4.2: Example of DOS program definition

Every time a scenario changes, traffic light programs have to be defined for every intersection which can be a cumbersome task to do for bigger maps. In figure 4.2 is a snippet shown which defines the traffic lights for DOS. And this is just for one standard four-way intersection that has to be defined or changed everytime a new scenario or map is created.

4.1.3. VTD

Although VTD® will only be used as an alternative viewer it has the capabilities to be used as a standalone traffic simulation as well as a set of different modules that are needed for traffic simulations. For creating a scenario in VTD® one can use the existing GUI and define everything that is needed. For the special case of traffic lights one would have to define and map every phase there is in a classic signal cycle (go, stop and the two attention phases) to each code so everything will be interpreted right. This step is necessary since intersections can get very complex and signal groups do not fall together naturally.

4.2. Traffic Signal Control

There are several different ways to control traffic lights more or less efficiently. But it all comes down to two big differentiations: adaptive and non-adaptive control. According to the Cambridge English Dictionary, adaptive means "'having an ability to change to suit different conditions" [3]. Therefor all controls that are not only time controlled can be described as adaptive, so it is a very ambiguous term. In the context of traffic signal controls 'adaptive' includes everything that does change behavior due to some condition change. Even a time controlled switch between different fixed-time programs would be considered

adaptive if these programs are designed to, for example, meet the requirements of morning and evening rush hours. Although this might be a contrast to what most people think of when hearing the words 'adaptive light control'. Mainly 'adaptive' gets confused with the term 'optimized' since it is often used for describing more complex control schemes. Also every reactive control is at the same time an adaptive control, but not necessary the other way round as the rush hour example indicates. But adaptive lights are necessary for modern traffic to flow efficiently due to the rising urbanization worldwide [12]. Intelligent traffic lights can be and are used to not only improve the traffic flow and therefor ease everyone's everyday live but also to ease the environmental stress that modern traffic creates [6]. Another strategies would be either a centralized traffic control that would direct the flow on a system wide planned basis [8] or a decentralized, intersection focused control approach [11].

5 Design

This chapter introduces the overall design of the solution and describes several requirements that will be fulfilled by it. An overview over the complete design can be seen in figure 5.1.

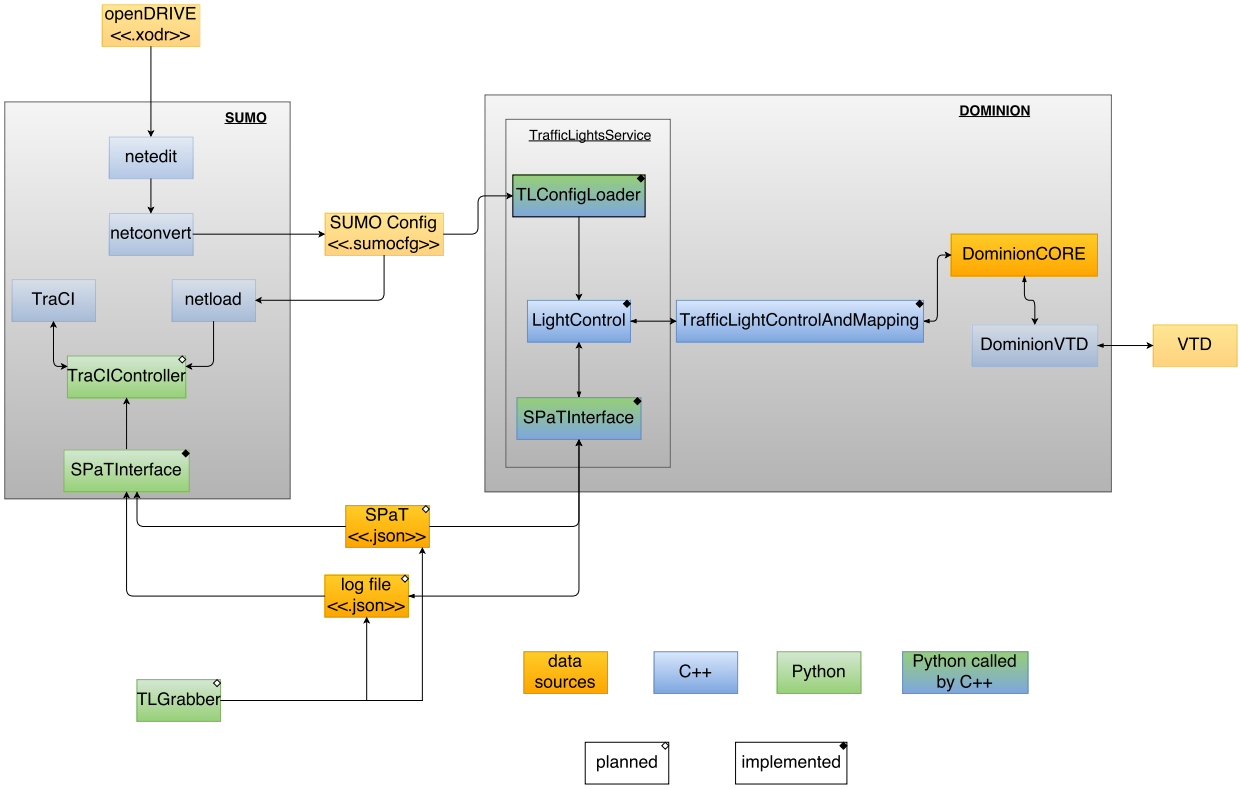


Figure 5.1.: Overview of the project

5.1. Requirements and long-term Design

Prior to the beginning of the thesis, some requirements have been formed due to long-term design or desirable features. Below, in list 5.1, are listed which global requirements were given (the solutions are presented at table 7.1 at page 35) in addition to the thesis problem, as well as the thesis tasks regarding the design.

Thesis Tasks

1. Design and implementation of an application that interfaces to all respective software modules based on netconvert[1]

[1] The naming of netconvert was a communication mistake at formulating the thesis problem statement, which was corrected in a verbal agreement.

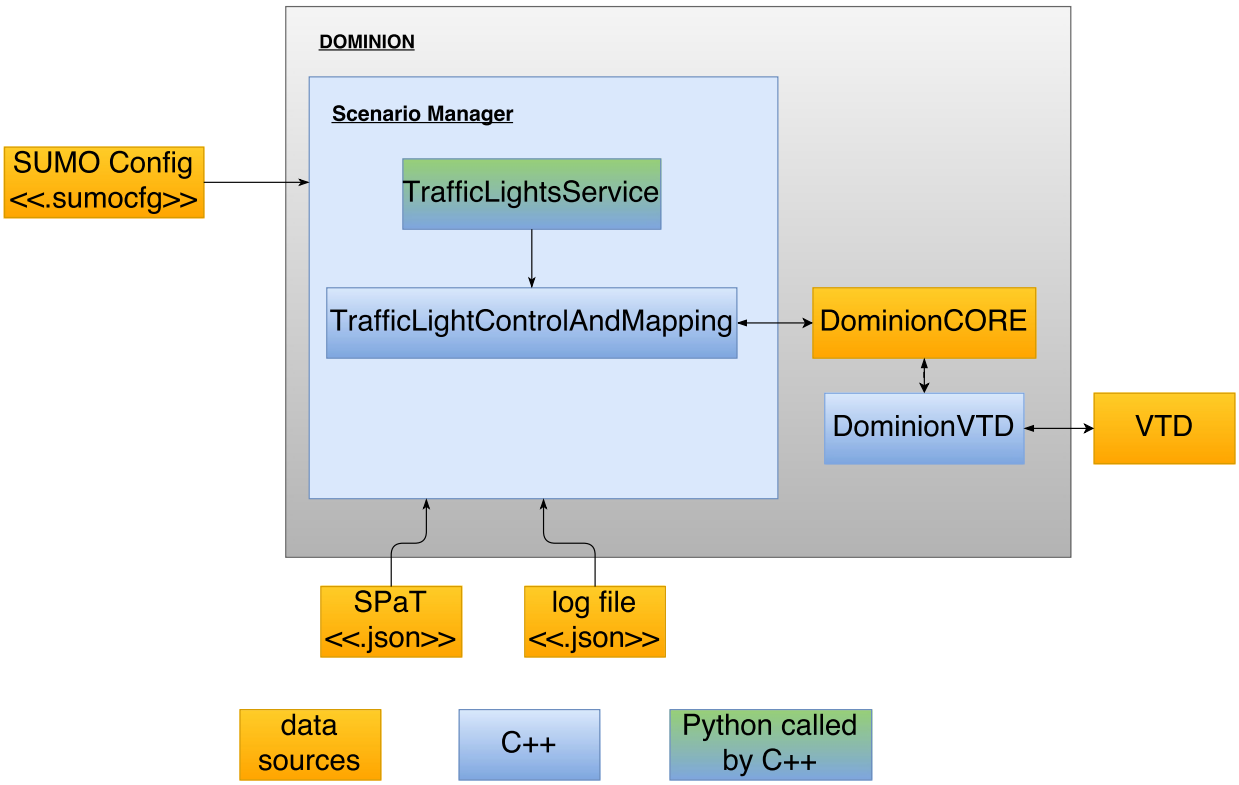


Figure 5.2.: Long-term integration

2. Enhancement of netconvert[1] to import recorded traffic light data and adaptive control schemes
3. Analysis of mapping of traffic light controls for high complexity intersections
4. Analysis of live-stream from real infrastructure
5. Analysis of hardware test for created algorithms

Additional Design Requirements

1. Features have to be provided as single, modular library
2. General method for traffic light transform and program generation

The here presented design is made for the long-term for not only being able to import SUMO-made fixed time traffic light programs but also to support different means of adaptive control mechanisms with a specific focus on public transportation. To achieve this, the programs written for this thesis will be integrated into the scenario manager of Dominion eventually, as shown in figure 5.2.

5.2. Fixed-Time Programs

To have working traffic light one has to begin with the most direct strategy for control, the fixed-time control. Hereafter, the methodology to realize this strategy is explained and shown how the new architecture supports this means of control.

5.2.1. Retrieving programs

Since SUMO's netedit already provides an approach to calculate and export traffic light programs into XML a program is in need to import these programs. The strategy of the TLConfigLoader is to decompose SUMO's output file format .net.xml to get all the needed information that DOS and VTD are needing to understand the information as well as being able to create a proper id mapping without hand crafted matching tables. As it can be seen in figure 5.3 in chapter6.2 the approach to retrieve programs is fairly simple and straightforward: Get every information that is needed and structure it in a way DOS can access it easily. Further in-depth details are written in chapter 6.2.

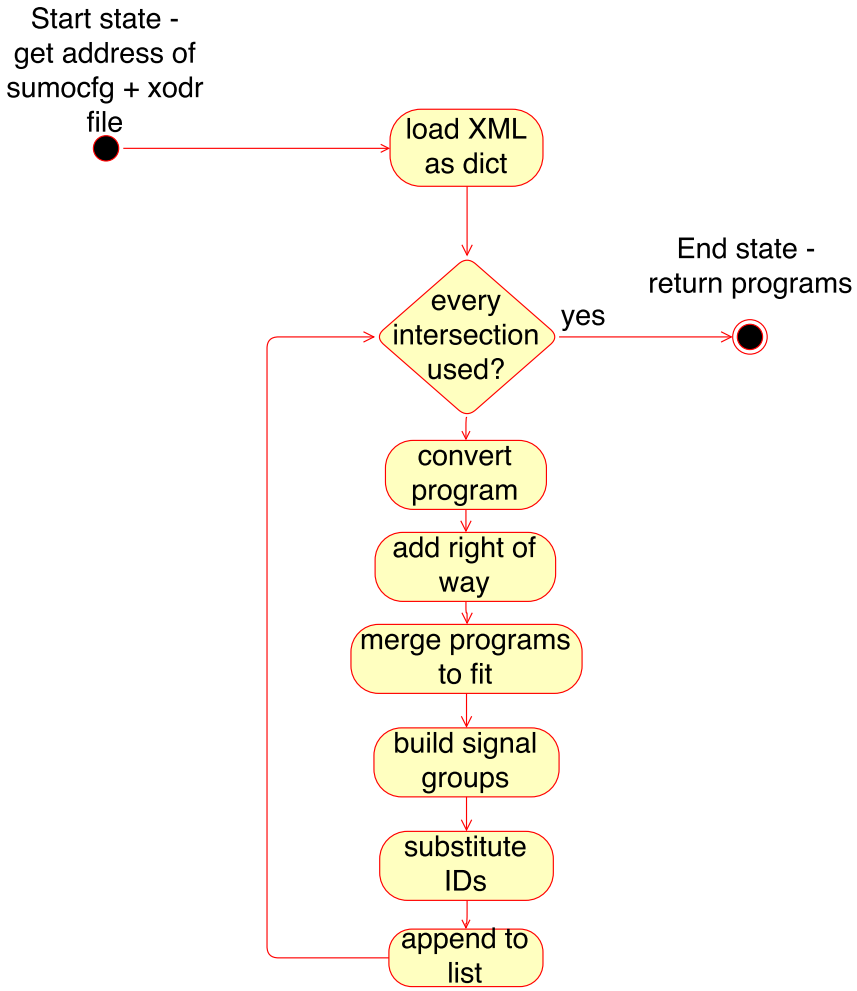


Figure 5.3.: TLConfigLoader sequence

5.2.2. Conversion Metric

One of the biggest problems was that SUMO and DOS use different structures for representing control light programs. SUMO uses a special representation as it treats every intersection with one phase for each path a vehicle can follow separately, where DOS treats only the signals that are defined in the openDRIVE specification. To map one representation onto the other a conversion was necessary. Figure 5.4 shows how SUMO interprets a

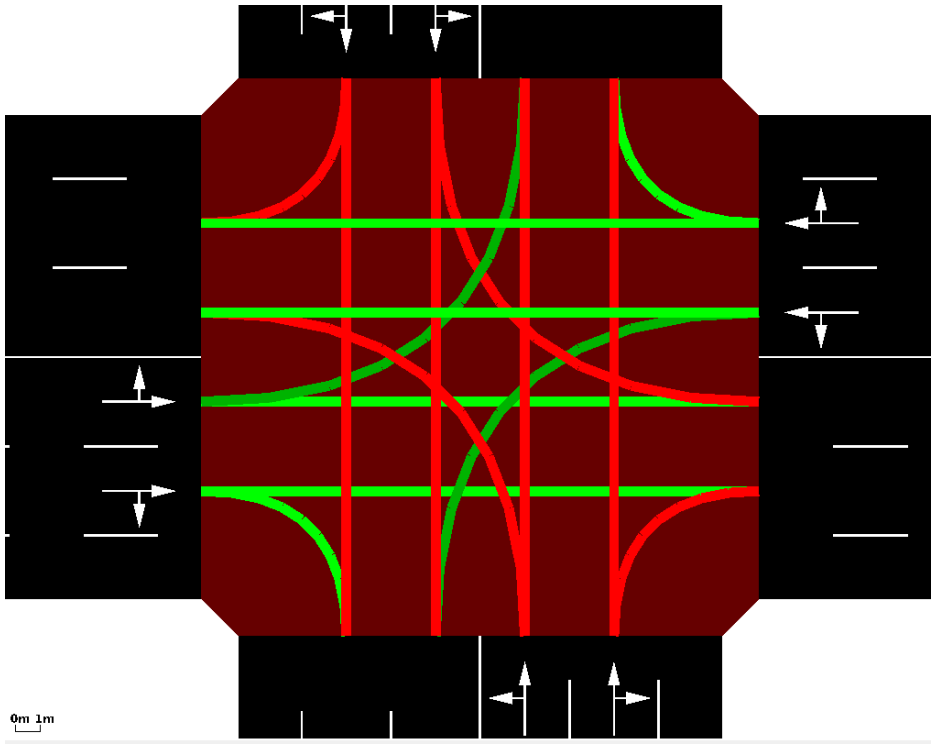


Figure 5.4.: SUMO's model of an intersection with its traffic lights

standard four-way intersection in the test track used to build this prototype. In this there are two traffic lights per lane per intersection, so eight signals in total per intersection that have to be controlled. Each of these pairs is associated with the allowed paths a vehicle can take to move across the intersection: it can either turn left or drive straight for the left lane or straight and turn right for the right lane. This results in four phase descriptions per phase per randomly created program by SUMO. To reduce this four programs to fit two traffic signals a merging method is introduced in the data mining process. Hereby the phases are merged pair-wise according to their respective lane of origin so that this programs can be associated with the correct traffic light. The automatic programs SUMO created showed that the respective left lane was treated as a left-turn-only lane, therefore needed a separate phase to be treated accordingly. This could be done by a slight modification of the merging method where now the phases are not merged two by two but three by one, merging all straight indicating phases with the one for right turners and thus leaving the phases for turning left untouched.
In figure 5.5 can be seen how the models are arranged for the presented design. This is similar to how DOS or openDRIVE itself sees traffic lights. The most important differences here are, that for DOS/openDRIVE the control structure is very strictly given by the map. That means each intersection has a certain amount of controllers associated with it, but these controllers are only associated with specific traffic lights they are responsible for. This also makes the creation and coordination of programs more difficult because the controllers can not be reassigned to new signals and vice versa, once the map is loaded.
For more complex intersections this conversion method could also be sufficient, depending on how the openDRIVE specification looks like. The biggest problem in mapping ev-

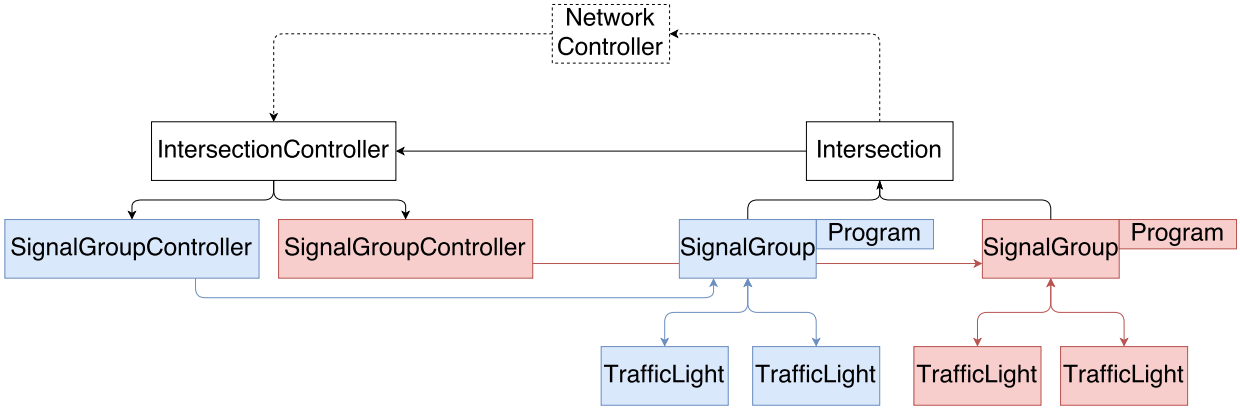


Figure 5.5.: Basic structure of models

erything together is the discrepancy in interpreting intersections. A big intersection with complex traffic guidance is very difficult to map and interpret as one entity, but should be possible to convert the same way as simple intersections when seen as superset of different parts of it. These parts would consist of subsets of already known intersections, which again could be realized with the shown architecture, as it does not depend on spatial information. To achieve a correct conversion and mapping there would be the need of an additional generator. Which would not only have the information from the openDRIVE file but also other geographical data to fuse with. This is no trivial effort to make, but necessary because the identification numbers vary from intersection to intersection, depending on who labeled it in the first place. Therefore, geo-data is key. After this mapping program is complete it could be build into the here presented pipeline or be used to generate an openDRIVE file that already has the real-world IDs.

5.2.3. Running programs

After structuring the retrieved programs comes the application of them to the simulation environment. Such a design should be as simple and straight-forward as possible by simultaneously being easy to use and to scale. Also should be kept in mind that in the long-run this project will be integrated into the scenario manager, thus it has to be easy modifiable to Dominion's structure. In figure 5.5 can be seen how this can be achieved. Basically it is a combination of the observer design pattern A and a variation of the model-view-controller design pattern (MVC) B to achieve efficiency and less error prone usage in terms of development. On a larger scale the design is based on the MVC (see Appendix B at 53) pattern with splitting the information holding model from the control of the system. With this split it opens up a very flexible application to various possible use cases. In the basic use case, a fixed-time program, the controller nodes will only pass through the simulation timestamp and return the current state for the attached program. However for applying adaptive strategies all there is to do is to add respective functionality to a controller class by e.g. enhancing them through inheriting the basic implementation. This way every basic way to control light signal systems as well as any adaptive strategies can be implemented and used. The view aspect of the pattern was already given due to the structure of Dominion. As mentioned on page 9 in subsection 3.2.2, Dominion is a model based framework built around a data core. Since this core represents the communication point for the whole network it also

represents an interface for various kinds of information and services. In this specific case it is the interface for the view part of the pattern, since all displaying applications get their input from there. The observer pattern on the other hand finds more application on the smaller scale. As to be seen in figure 5.5, the model and the control part are built to depend tree-like on the inside. With this it is possible to create a very slim and manageable data flow for each strategy independently. On top of that this combination of patterns creates a system that can act like a state machine and therefore can be very predictable. Also this nodal structure can be directly taken to own threads and thereby can directly be paralleled on hardware, which would result in an even closer proximity to reality. All that together makes this design very efficient since it nullifies the need for other re-implementations in the future. With such a stable base structure it would even open up the whole testing process to hardware-in-the-loop (HIL) test scenarios, which are described in chapter 7.2.

5.2.4. Connection to Dominion

As stated in the requirements to begin of this chapter, it was given that the whole project should exist as a library for later inclusion purposes. therefore a connection to the Dominion core is needed. This will be established through another interface to separate the Dominion domain from the traffic light control domain. The advantage of this approach is that the existing scenario manager will be able to just include the library and provide the respective information to it to use its design. This has also some disadvantages though: Due to this anomaly by design in terms of Dominion applications (and other reasons, see 5.3) the implementation of adaptive strategies will have to be postponed for the sake of a clean integration and lower integration costs.

5.2.5. Fixed time control strategies

To implement fixed time strategies several abstractions have to be considered. First, the program itself (see definition at page 3) and what model it should be associated with it. Also the behavior of all abstraction layers should be considered. This is solved by attaching programs directly to the signal group since they are the lowest tier element that has to be manipulated for fixed time controlling. The justification for a seemingly irrelevant representation of single traffic lights is, beside the technical need for storing the affected signal's ID, discussed below in section 5.3. These programs will function as a look-up table, where the states of a signal group are mapped to the respective simulation time, details are described in chapter 6.

5.3. Adaptive Programs

As stated in section 4.2 most modern control systems in urban areas are adaptive. With the presented architecture both forms will be realizable, due to the separation with the MVC pattern.

5.3.1. Retrieving Programs

To keep things consistent, the basic use case to create an intersection that is adaptive controlled should be through SUMO. This can directly be done through setting a keyword per adaptive strategy that will be recognized in the parsing process and then build the respective versions of controllers for that strategy. It should be done through a generator

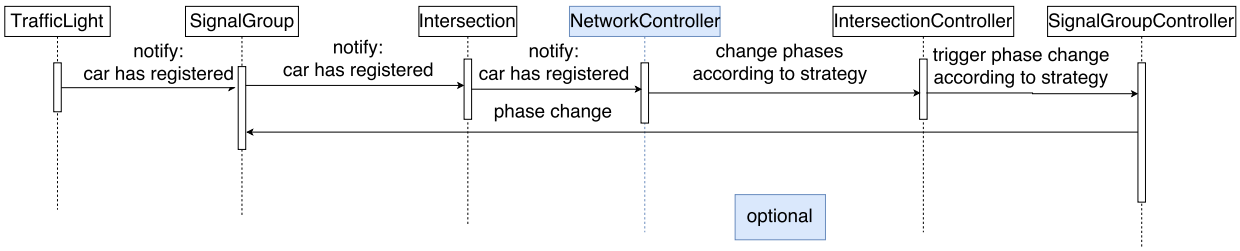


Figure 5.6.: Flow of an adaptive signal control

interface in the future which is only designed but not yet implemented. Ideally this generator should re-use code already written to implement said strategies in SUMO, practically this wouldn't work without restructuring the existing strategies to a more abstract format so that both different simulators could work with it.

5.3.2. Running programs

When everything is set up according to the respective strategy, all sensors should be interfacing with the TrafficLight class. When this is given, the communication flow for such a strategy would look like in figure 5.6 All adaptive strategies have some key values that they are built around, be it the amount of cars, time they need to pass an intersection [4] or other metrics like the general daytime. In the example in figure 5.6 a mechanic is supposed, that features the absolute amount of cars in front of the the traffic lights. Here can be seen that in the first step the change of the key value is propagated through the structure to the deciding tiers, intersection or network respectively. Again, this architecture allow a connected or distributed way of controlling traffic lights/intersections. According actions will be commanded on base of the respective strategy. Suppose a strategy where the arm of an intersection that has the most cars gets its go phase stretched up to a maximum that was pre-defined. With this structure the researcher has all the freedom needed for experimenting with different approaches. If the additional time slabs should only be distributed once all signal groups went green, then this only has to be implemented into the network controller. If it should be direct dynamically, then that is possible too by not building in any timing constraints and just directly allocate extra time.

5.3.3. Adaptive time control strategies

In contrast to fixed time programs, most non-trivial adaptive strategies are much more complex. There are many ways to control an intersection to call it adaptive, but all of them have in common that some kind of sensor is involved. These sensors will have some connection to the traffic lights, be it indirectly through the road they control (e.g. a an induction loop in the asphalt) or directly like a camera, evaluating the traffic in front of the light signal. Another, more realistic, approach would be to just add another Observer pattern to the traffic lights. They would function as observers of vehicles that are passing them, to simulate real-life radio technology. Every vehicle could register itself at every traffic light they are passing and stay registered as long the communication can be kept alive. To create an interface for future implementations of numerous of adaptive strategies the basic design does feature a model for traffic lights. This model is already considered to be observed by the next higher abstraction layer, the signal group. This way all sensor infor-

mation can eventually be lead through the model to the decision making levels, like the intersection or network level. Again, because of the variety adaptive controlling can come in many different forms the MVC pattern will show its elegance and practicability. Due to the pattern, every control command has to be implemented in a controller. That does not only ensure a certain stability in the development process, because all the observers are already set up, but also a very high flexibility. Also, by enhancing the basic progressing for fixed time controls, every type of strategy can be fitted into this architecture. For example by implementing a stretching method, that will stretch the go phases by simultaneously keeping the maximum program length, one could create a simple, actuated light control. High level strategies could be realized by directly implementing a parallel control stream to the fixed-time stream that works for example reactively according to the special rules of the strategy.

5.3.4. Special Focus: Public Transport

As the title of this thesis indicates, the presented methods are also made for public transport favoring strategies. The basic idea for every strategy is that via some sensor the existence of public transport vehicles at the intersections respective arms will be recognized and passed to the controller. This can also include taking schedules into account rather than responding reactively. This sensor, representing vehicle-to-infrastructure (V2I) communication, would signal the system the count of present public transportation vehicles. A straightforward extension to many strategies would be to react in a much stronger way if there are public transport vehicles on the lane, for example they could be counted as 10 cars at once to keep the strategy analogy. Or they could add a multiplicator to the delayed time in [4] that would inflate the priority of said lane. In reality this could lead to a randomized deadlock solution if a bottleneck intersection has multiple buses for example arriving at the same time. This could be solved by also taking main roads and statistical data about usage into account.

5.4. Interfacing the Reference Track and Research Intersection

For interfacing with the research intersection some problems have to be solved:

1. Prepare for future possibility of live stream
2. Use as many already established interfaces
3. Be as lightweight as possible
4. High usability

Originally the idea was to create a live mirror into the virtual Braunschweig for not only having a complete overview over the net state of the Reference Track but also to be able to conduct studies under real live traffic light conditions. This is unfortunately not possible yet due to hardware and connectivity. But since the bigger plan is to achieve this one day, it would be the best to already have the basics implemented. Although a live stream is not possible yet, there are already many interfaces, for example a RESTful interface for querying states of whole intersections on the Reference Track or an interface to a database where conducted data can be stored. The purpose of software that interacts with the Reference

Track should also be to add value to the whole project by integrating properly with existing systems. Additionally it should be lightweight to minimize time spent on increasing quality and to keep it as maintainable as possible. Furthermore, a lightweight software can better be deployed on single-board computers like the Raspberry Pi, which would cut down the economical costs for usage and benefit the research budget. And lastly this software should be very usable by offering some comfort functionality out of the box, like filtering the data already so the use cases will rise further.

6 Implementation

This chapter explaines the implementation of the prior presented design and the reasoning behind several design decisions.

6.1. Prerequesits and Disclaimer

To the date of handing in this thesis the presented work is programmed and tested with Python 2.7 and compiled with Microsoft Visual Studio 10. Due to the fact that Visual Studio 10 is already 6 years old there had to be made some compromises in terms of using modern C++ techniques. Furthermore only the fixed-time approach is implemented since there were several cut backs in terms of features that could be built upon that would be needed for an adaptive control system with this design.
The presented prototype is by no means complete or bug free. It is based on the track "1310_Urban_v4.xodr". This implementation is a proof of concept and the beginning of a bridge program with greater focus on functionality than being well tested.

6.2. TLConfigLoader

```
{
  "configFile": "D:\\Dominion\\User\\Libraries\\
     TrafficLightsService\\python\\1310_Urban_v4d.net.xml",
  "programs": [[{
    "intersectionID": "10"
  },
  [
  [
    {
      "foes": [2,3,5,6,7],
      "phases": [
      [2,35.0],
      [4,10.0],
      [4,35.0],
      [4,10.0]
      ],
      "response": [2,3,6,7],
      "signalID": 242
  },
  {
      "foes": [1,2,3,6,7],
      "phases": [
      [2,35.0],
      [4,10.0],
```

```
      [4,35.0],
      [4,10.0]
      ],
      "response": [2,3,6,7],
      "signalID": 236
  }]
  ]]]
}
```

Listing 6.1: Returned object from TLConfigLoader

The TLConfigLoader is written in Python and it is called from LightControl which is written in C++. This enables the use of Python's excellent features for XML, JSON and string manipulation, which are much more usable designed than the typical C++ parsing libraries. To achieve this linkage, boost's Python library is used which in general provides a wrapper around the Python.h but is very wide known and tested.

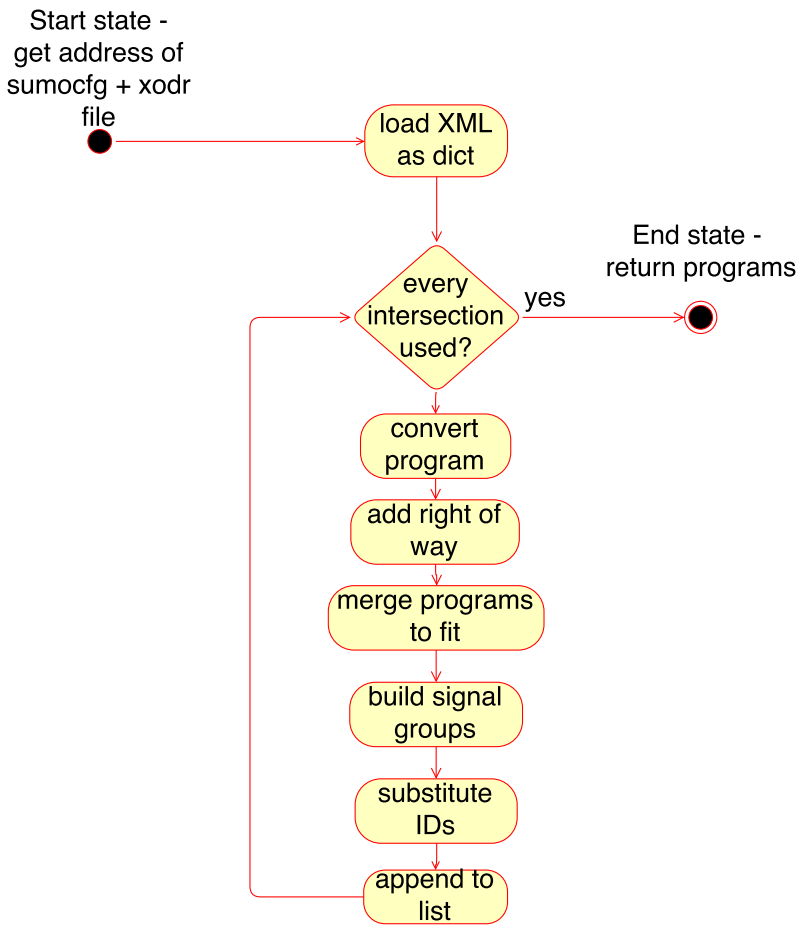


Figure 6.1.: TLConfigLoader's workflow

The data object shown in listing 6.1 is built in several layers throughout the data mining process, which is depicted in figure 6.1. To implement this process, a class-less approach

was taken due to the specialty of the problem and the requirement to proof a concept, rather than to build production level software. All these methods, shown in a Python-like style, in figure D.2, are transforming a Python dictionary in a way that it gets a clear structure, as listing 6.1 shows.
Nonetheless, the TLConfigLoader is separated from the main program through an interface which can be found at figure D.7 at page 62. This was chosen for future modifications and improvements sake throughout the refactoring process to separate concerns and enable the responsible developer to delegate several parts of the system to other team members if needed. As shown in 6.1, the workflow of the program names explicitly the step of merging the programs, whichs necessity is explained in sub-subsection 5.2.2 on page 19. This step is by no means to be understood uniform since real-life traffic networks are far more complicated as the artificial track that was used for the development of this prototype.

6.2.1. LightControl

LightControl is the heart of this thesis. There are two ways to use it, either with fixed time programs or with adaptive programs. For the sake of the adaptive control the storing and controlling is split in different classes. This has the advantage of clear encapsulation and therefore of better maintainability.

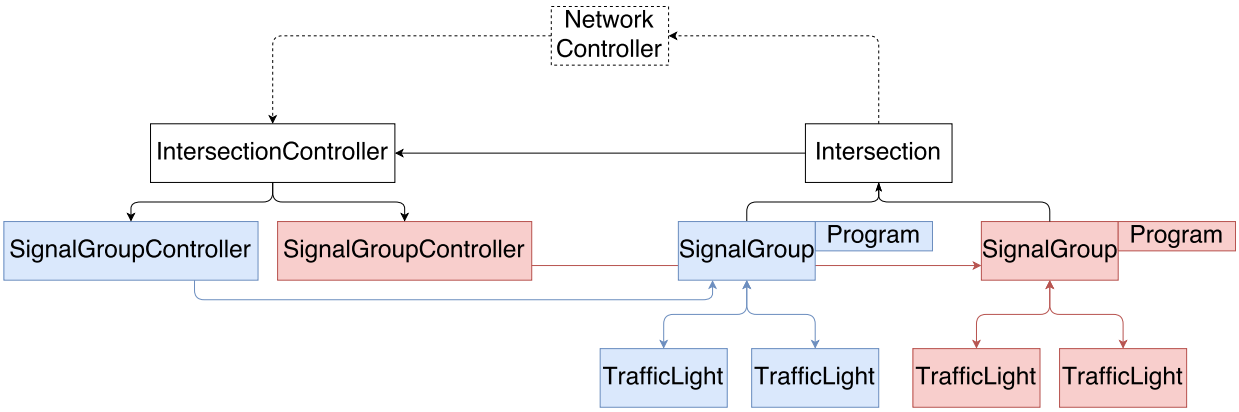


Figure 6.2.: Control loop inside LightControl

In figure 6.2 is figured how the control of traffic lights is organized. As stated the control is split from the data representation, former on the left, latter on the right. When talking control of light signals one has to focus on the signal groups since they are the lowest mattering level of abstraction, that is why the TrafficLight entities do not have an own controller.

Structure

At the top level might stand a network controller which has the overview over the complete traffic net with every intersection. This is only necessary for controls like Balancing Adaptive Network Control Method (BALANCE) and Method for the Optimisation of Traffic Signals in Online-Controlled Networks (MOTION) [8][11] that aim to optimize on a network level. With or without the network control, a traffic networks signal state can be described as a set of intersections. Each intersection has an intersection controller attached to it which can give commands to each signal group controller held by it (see D.1). These signal

group controller form an interface for manipulating the state of a signal group from which only one is held respectively. A signal group is basically just a list of traffic lights that have the same state at the same time, thus follow the same program. But due to the architecture of Dominion and specifically DOS, only signal groups are important for interfacing. Signal groups belong to a specific intersection and do hold traffic signals which are belonging to them. Those traffic signals are representing interface for future usage in adaptive control strategies. At the moment of this thesis the only way to implement them already would have been to tear open the existing interface and nullify the effort to encapsulate everything in a separate library.

Features

Each intersection can get the information from each signal group what its state is and what the next phases are - which may be interesting in overall control. In general does every upper storage object have access to the information of its lower elements, so a traffic light is accessed by its signal group which can be queried by its intersection and that can be read out by the net control. The advantages of this design start to show when moving from fixed time programs to adaptive methods. Where in the former case all 'calculations' are made by the programs themself, compare figure 6.3, a more advanced scheme for passing information is in need when talking about adaptive control. For this use case each data object is either a subject or an observer according to the observer design pattern. Every value calculating entity is being observed by its higher up so all necessary information is available at every level it has to be. Through this daisy-chain a control loop can be established like shown in figure 6.2.
Technically these links are realized with dereferenced smart pointers to keep everything tied together beyond the initialization and at the same time being able to use everything as simple objects. This step has been made to increase the maintainability of the project even by programmers less experienced in C++. The programs themselves are representing a mapping data structure which only needs to have the simulation time an will return the current state. All details about the current phase, what phases are in the program and how long each phase lasts are stored as class member variables. The decision to encapsulate it like that was made to keep the adaptive programs separated from fixed time programs, respectively to be able to build up the complexity of programs from simple to complex. With this default implementation, more complex cases like actuated control [4] only needs minor enhancements to realize.The class diagram can be seen in figure 6.3 or larger in the attached class diagrams at page 59.

6.3. SPaTInterface

The Signal Phase and Timing (SPaT)-interface provides access to other sources than fixed programs or known adaptive methods. By the requirements of the thesis problem it mainly shall interface with the data that will be recorded from the traffic lights on the Reference Track but it can be imagined as an interface to other sources. It is held very simple and relatively similar to the interface for the library that controls the simulated traffic lights, so that these can be merged through a refactoring process.

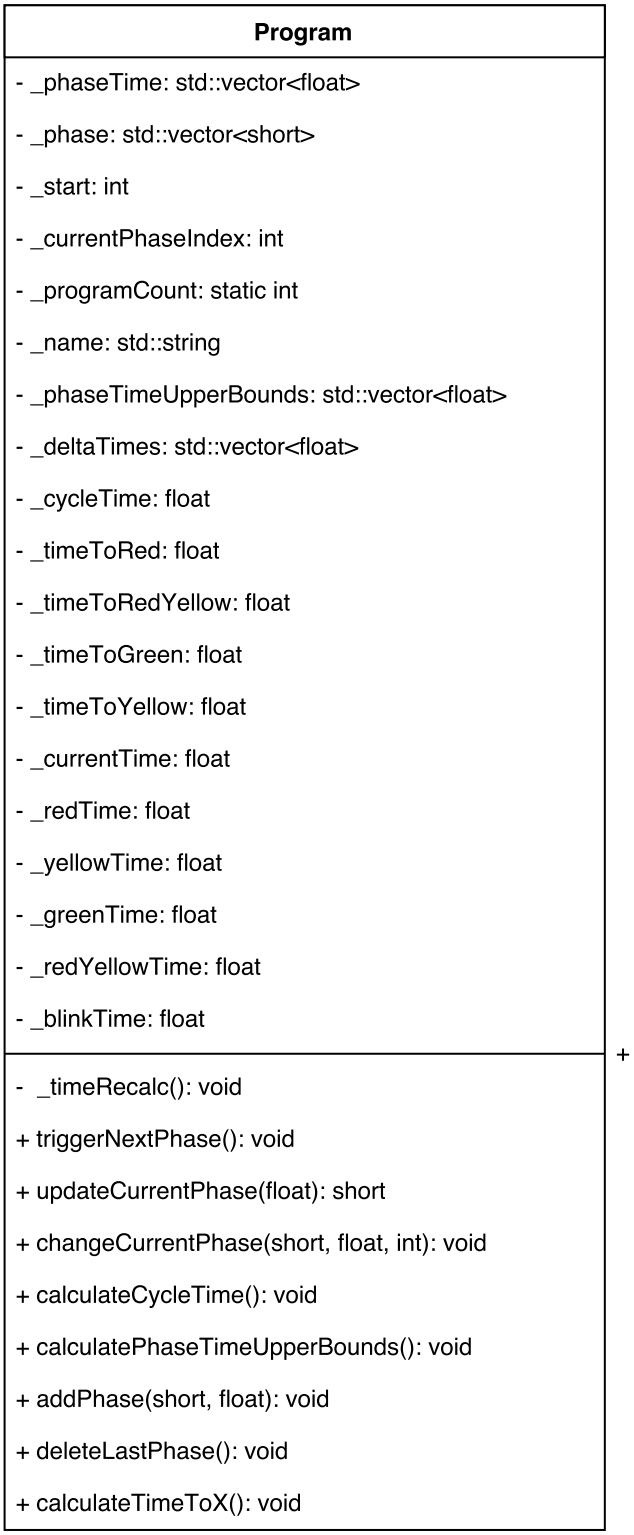


Figure 6.3.: Traffic light program

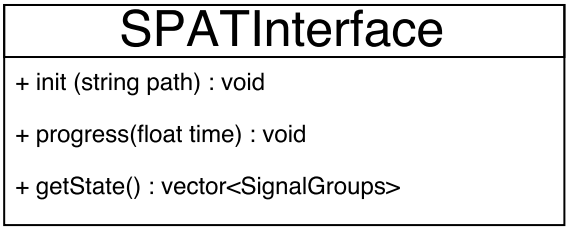


Figure 6.4.: SPaT interface

6.4. TLGrabber

As stated in the requirements in the beginning of chapter 5 and especially in section 5.4, future plans contain the possibility of digitizing the traffic lights of Braunschweig to access the original phase changes from within the simulation environment at DLR. To achieve this within the given possibilities the TLGrabber was created, a tool that allows to log the already accessible traffic light states.

6.4.1. Requirements

The TLGrabber's goal is to create an easy to use logger for the SPaT messages the RESTful interface from AIM provides. Among the feature of logging the messages in a given time period it was additionally asked for the feature of filters for the information provided. For example only one field of information for every signal at the intersection or just to get one traffic light specifically. Beside these requirements from the workgroup for AIM, a need for lightweight implementation arose out of the prototype nature of the project.
To achieve the lightweight of the implementation it was decided to write the TLGrabber in Python. This brings not only the advantage of a wide set of tools for handling web requests but also a very easy-to-use way to handle JavaScript Object Notation (JSON) data, which is basically the format provided by the RESTful interface. Although other languages also have these advantages, Python was chosen because other tools for the Reference Track are already written in Python and therefore it would be easier to share the maintenance work on the tool later on. Basically the TLGrabber enhances the RESTful interface that is already given. It creates the possibility to log the states of a given intersection and provides a method for reading in the data from a log file. Since it is written in Python it is straightforward to create a new data configuration and read it into C++. It furthermore represents a hook point for the implementation of replaying logged SPaT data in the future, but has not been implemented yet because the interface itself was announced to get a major rework in the future. This rework might render the work useless and therefore it is postponed until then. Furthermore should the implementation be designed as close to the TLConfigLoader which will also get a rework in the future, due to already formulated, new requirements.

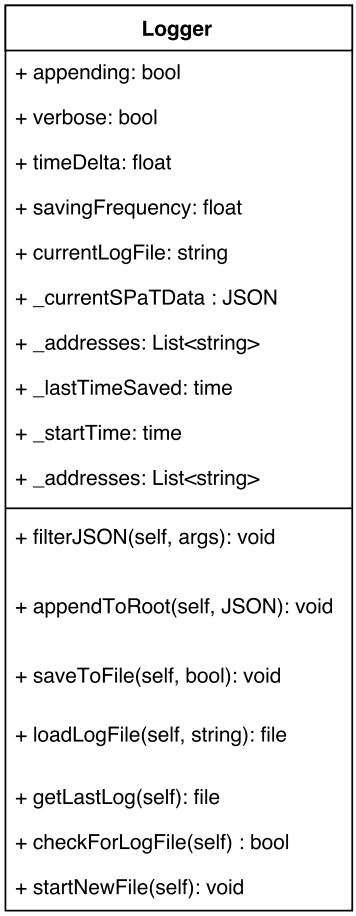


Figure 6.5.: TLGrabber's Logger class

7 Evaluation

7.1. Meeting the software requirements

Requirement	Problem	Solution
Live-stream of Reference Track traffic light signals	Network infrastructure	Logging
Program as library	Dominion Structure	Refactoring and Integration in ScenarioManager
Adaptive traffic light controls	VTD source not accessible, DOS lacks important features whichs implementation would exceed the time for the thesis	What can be implemented, has been implemented. Further features and details are presented in design
General method for light transform	SUMO/Dominion have different approaches to realize junctions	Abstract interface for import

Table 7.1.: Requirements and their solutions

This table is an enhancement of the list already shown in chapter 5. Some of these requirements could be resolved while others had to be left out. At the moment, the mode of operation is focused on Car2X communication for traffic light system states. A live-stream of the states of all intersections to the backend infrastructure was not planned in its design up until now. Because of the limited network capacity it can not be realized at the moment. Therefore, the TLGrabber prepares this feature in the future by implementing a recording functionality with already maintaining the needed interface for later integration.

Live-stream of Reference Track traffic light signals

While investigating the infrastructure of AIM's Reference Track's infrastructure it became obvious that a real-time live-stream is not feasible at the moment. The problem is that the Reference Track itself is still in a state between prototype and fully operational research facility in terms of interfaces, hardware and network speed. Especially due to the last point, the network speed, it is not possible to achieve a live-stream, even delayed because the network would collapse at this high request rate. Furthermore the complete Reference Track and thus also the Research Intersection would be blocked for the time of those experiments. To reduce this and to use what is already given, the decision was made to record the phases and create logfiles of them and use them later to play them back into the system. This is done by the TLGrabber program, shown in section 5.4 at page 24 for design details and section 6.4 at page 32 for implementation details. As for the playback part, the interface was defined in section 6.3 of chapter 6 in figure 6.4, which fits to the implementation of LightControl (see figure D.4 at Appendix D).

Program as library

To design the whole project as a dedicated library was a requirement formed out of the plan to integrate this thesis' work directly into the scenarioManager of Dominion. This was achieved, the main part is separated from the Dominion application workflow through an interface. Background to this is to have everything regarding basic functionality of the simulation in one application, so that the scenarioManager really does manage every aspect of the scenario. That is also why there are some pieces left to be refactored at a later time, because with this upcoming integration task will also come a general refactoring and a discussion about what should be included and which features can be skipped or altered in a refactored program. It would be inefficient to tailor everything to this interim solution rather than to aim at better quality in the program that will be used in operative business.

Adaptive traffic light control

To create a viable, design-conform way to integrate adaptive strategies to the systems, there are several obstacles to overcome. First, every adaptive strategy that is more complex then merely switching the fixed-time programs according to daytime or weekday, fitting statistical data gathered about the very intersection, has to interact with the environment in one way or the other. The intersection needs some kind of load information from which it can derive the next most important road to get a "'go"'-signal. To enable this communication, there has to be a connection of the data model of the intersection and its submodels to the environment or the vehicle model itself. This would contradict the requirement formed above, to create the program as library to integrate it later into another program. With this requirement the whole architecture was separated from Dominion's data management through an interface. If the decision had been made to realize this functionality while considering this requirement it would render the interface too large to be qualitatively pleasing. It would result in a very unclean solution that would create more work to clean up due refactoring then it would be to do it right when the time is come for the refactor to occur.
In addition to that, there is another problem present from a quality and maintenance point of view: Since several teams are working on different aspects of traffic research, several models are existing multiple times. This is also the matter of fact for traffic lights. Therefore, the real implementation of the feature to actually insert adaptive traffic light strategies would only create more work to be done in the long run. This would do more harm than good in terms of maintainability and code quality, as it would only add another item on the already long list for aspects that has to be thought of for an indeed needed refactoring. This is why in creation of this thesis it was decided to do the design work only then also implementing it. Another crucial fact that arose while researching this problem was, that several features that would be needed for a successful communication are not implemented yet or the relevant parts of the code are not accessible.
Since VTD® is a framework created by a third party vendor, there was no chance to get the source code to implement the features. To request them to be implemented by the vendor VIRES would probably taken longer to bargain than there was time to finish the thesis. And although the source code was available for Dominion, the right insertion places would be so deep in the core of the framework that that alone would take at least the participation of one core developer to implement it efficiently. As mentioned in subsection 5.3.3 at

chapter 5, one possibility to imitate real-life solutions would be to let the vehicles register themselves at intersections they approach according to which lane they are on. This would need several orientation features based on the vehicles position in the virtual world, among them a method to traverse the road specification towards traffic lights/intersections and mapping that to the geometry of the map to be able to consider obstructions some cities might have. These features do not exist yet and to create them would also need developers of Dominion to assist as it is a large code base with a myriad of dependencies to it. To learn alone all those dependencies and then add those features would almost eat up the whole time budget for one master's thesis. Therefore it was also postponed to implement later, when those features are available to avoid doubled workload.

General method for light transform

When it comes to road designs it is highly dependable how the city around those roads came to be and in which country they are in. As one would find more 'naturally' grown cities in Europe, evolved for several hundreds of years to fit the transportation needs of past times concerning horse carts and pedestrians, more artificial planned or preplanned cities can be found in other parts of the world like the United States of America. With this kept in mind, it becomes clear to see that a general correct parsing for intersections is no trivial task. Therefore, an approach was taken to leave the perfection of this open for the future to be achieved.This was done by creating an interface in the parsing process to create a special intersection classification in which a mapping of geographic data with the road specification will take place.
Another difficulty was to pair and map the two highly different representations that openDRIVE (and therefore VTD®), Dominion and SUMO are using for their road and traffic light simulation. SUMO was used as a starting point which created the following discrepancy: SUMO is treating traffic lights path-wise, so that every path a vehicle is allowed to take at an intersection gets its own phase mapped to it. This means it does an internal remapping from openDRIVE data. VTD® and Dominion on the other hand are using openDRIVE's mapping and structure natively. Therefore a remapping of the traffic light programs was necessary, which takes place at the TLConfigLoader program. This method is created for parsing standard four-way intersections in the first place to create a starting point for a prototype to enhance according to the maps and special types of junctions that will have to be parsed in the future.

7.2. Evaluation of hardware requirements

After asserting that a simulation works as specified, the next step would be to bring the test to the hardware in the field. The problem with this step is that there is no standard for the hardware that controls traffic lights. Although the assumption that often some kind of programmable logic controller (PLC) is involved is a good guess, the real setup will differ from country to country, city to city and maybe even between districts in the same city. This is depending on when an intersection was maintained the last time and which method of control is implemented and how the service contract is structured. Because of this big variance between possible hardware specializations it would be impractical to build several test setups for different compositions in the long term since they are likely to change often in the long-term. Therefor, to check the compatibility of program and hardware also the hardware should be simulated before rolling out a new control mechanism to the streets

for a live test.

7.2.1. Simulating Hardware

Hardware simulation can be done through modeling libraries like SystemC or languages like VHDL and Verilog for example. While the latter two are directly applicable to hardware like Field Programmable Gate Array (FPGA)s, the former has to be translated first to put it directly on the hardware. But since SystemC is a software simulation of hardware components, it can be used before making any purchases of special testing hardware. Therefore, it is more cost efficient to start with it since, as library for C++, it can also benefit from the myriad of development tools that exist for it, such as automatic test generation or code analysis tools.

Although that is not a trivial task to do , continuous integration with all its features can be established [14]. With this powerful development technique it would be possible to create virtual hardware simultaneously to the software for the traffic light as part of the software quality pipeline. As another part one could envision an automated design space exploration step in which optimized hardware could be simulated for said software. Nonetheless, there have to be verified TLM models for the respective RTL models used in said hardware configuration, but then the optimized hardware configuration could even be made deployable to FPGAs for a real live test.

Furthermore, testing on virtual hardware gives developers the great opportunity to test their software for fail-safety. This could cut costs in the long run since real hardware has the feature to wear out in the testing process, where virtual hardware could be tested figuratively forever on a matching computer system. Also those hardware tests could be, again, automated to save time and money. For example if one were to use real FPGAs to test the implemented software every time not only the time of the tester would cost money but also the FPGA test boards themselves are not cheap either.

8 Conclusion and Outlook

The purpose of this thesis was to present methods for integrating control algorithms of light signal systems for various simulation environments with special consideration of public transport. To achieve that, a design was proposed and discussed why it solves as a good way to approach the problem of representing a general architecture for light signal system's control. Therefore, the general systematic background, as well as the theoretical and historical background of traffic lights was presented and shown what the state of the art at the moment is. Afterwards the design proposal was made with said discussion, followed by describing the implementation of this architecture in more detail. Furthermore there was a discussion how to expand the simulation frameworks for hardware evaluation. Also several features for this domain were prepared, as there would be full adaptive strategy support, general mapping features and a full mirror of the Reference Track's states.

8.1. Outlook

A thesis rarely marks the end of a problem and as a nature of software development there is always more to do and to improve. Here are some improvements that should be made in the future.

8.2. Code Improvements

As this thesis provides only a functional prototype there are some architectural improvements that could and should be made in the future to develop a maintainable system. Furthermore there has to be an enhancement of the implemented prototype to achieve its full potential when circumstances are given to do so.

8.2.1. Functional Improvements

The presented prototype is merely a proof of concept and not ready for operational use yet. To fit the task to the circumstances while creating this thesis, it was not possible to include some features, also some feature requests arose throughout the development as some others had to be discarded for now.

Generalization of traffic light conversion

As mentioned in chapter 6, the prototype is built upon a specific track as basis. This specific track only consisted of one type of intersections, the four-way intersection. To be able to enhance the merging functionality at a later time, the merging process was designed separate from the rest of the data retrieval in the first place. This generalization has to be refactored in a way that it not only detects the type of intersection it wants to convert but also to be able to handle very complex intersection situations. Ideally would be to find a fully automated solution that works with every type of network crossing but for usability's sake, it would be an advancement already if the standard types were detected correctly and a human has to do the mapping once for the map through a GUI per hand.

8.2.2. Design Pattern Improvements

MVC Pattern

The MVC pattern could be enhanced by altering it in a way that the prototype can make use of different hardware to control single intersections. Therefore the controller would have to run on separate hardware units to approximate reality better and be able to verify the to-be-done hardware simulations.

Observer Pattern

The prototype is mainly the Observer design pattern A in its basic form for fixed update methods. This could be improved by remodeling it in a way that it accepts callback functions to improve the usability of the structure. This way it would be easier to develop new types of controllers with different update methods for example.

Introducing a Singleton

The network controller should become a singleton to prevent future mistakes in instancing and using the system. Right now it's still allowed to (theoretically) have multiple network controllers. Although this might also be interesting in simulation (imagine traffic light systems that should run independent from each other) this could also be achieved by introducing a new type of controller (for example a sub-system controller) above the intersection controller and under the network controller. A singleton would add more consistency to the data objects.

8.2.3. Refactoring

General

Refactoring has to be done before there can be any other further development on this project. Since this is a one-man system, there are guaranteed points that aren't as precise defined as they could be and therefore need at least as second pair of eyes to look over it.

TLConfigLoader

The TLConfigLoader should be modified so that the programs can read out and validated through a regular expression. This would not only simplify the validation process of programs but also make it easier to check for off-the-norm patterns for special experiments.

Data management

At the moment the data is managed in another way than all other applications do at DLR - this probably should change. The thesis prototype has to be included into the scenario manager and be adapted to the normal data flow.

Bibliography

[1] BBC. The man who gave us traffic lights, July 2009.

[2] Lance Day and Ian McNeil. Biographical Dictionary of the History of Technology. Routledge, New York, 2002.

[3] Cambridge English Dictionary. adaptive meaning in the cambridge english dictionary, 2017.

[4] J. Erdmann, R. Oertel, and P. Wagner. Vital: A simulation-based assessment of new traffic light controls. In 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pages 25–29, Sept 2015.

[5] M. Fischer, A. Richter, J. Schindler, J. Plättner, G. Temme, J. Kelsch, D. Assmann, and F. Köster. Modular and scalable driving simulator hardware and software for the development of future driver assistance and automation systems. In Driving Simulation Conference, 2014.

[6] B. Friedrich. Strassenverkehrstechnik, volume 58, chapter Koordinierung von Lichtsignalanlagen in staedtischen Strassennetzen - Bewertung der Umwelteinwirkungen / Offset optimization of traffic lights in urban road networks - Assessment of environmental impacts. Forschungsgesellschaft fuuer das Strassenwesen (Germany), and Bundesvereinigung der Strassenbau- und Verkehrsingenieure. 1957., 2014.

[7] VIRES Simulationstechnologie GmbH. Vtd details. `https://www.vires.com/docs/VIRES_VTD_Details_201403.pdf`. Accessed: 23.08.2016.

[8] G. Kruse. Motion netzsteuerung - optimierung der lichtsignalsteuerung im einsatz. Steuerung kommunaler Verkehrsnetze, pages 37–52, 2003.

[9] Stefan Lämmer. Reglerentwurf zur dezentralen online-steuerung von lichtsignalanlagen in straßennetzwerken. 2007.

[10] C. Mcshane. The origins and globalization of traffic control signals. Journal of Urban History, 25(3):379–404, Mar 1999.

[11] J. Mück. Balance: Adaptive lichtsignalsteuerung für straßennetze und einzelne knotenpunkte. Steuerung kommunaler Verkehrsnetze, pages 53–66, 2003.

[12] Department of Economic and United Nations Social Affairs Population Division. World urbanization prospects: The 2014 revision, 2015.

[13] DLR Institute of Transportation Systems. Simulation of urban mobility.

[14] David Spieker. Development of a continuous integration uvm-systemc framework. Master's thesis, Technische Universität Braunschweig, 2016.

Listings

4.1. Example of SUMO program definition 13
4.2. Example of DOS program definition 13

6.1. Returned object from TLConfigLoader 27

List of Figures

2.1. Research Intersection in Braunschweig (Source DLR (CC-BY 3.0)) 4

3.1. Overview AIM 5
3.2. VRLab 6
3.3. MoSAIC Setup[5] 7
3.4. Main part of the Reference Track (green line) (Source DLR (CC-BY 3.0)) . . . 8
3.5. Dominion Overview (Source DLR (CC-BY 3.0)) 9
3.6. Model based design of Dominion (Source DLR (CC-BY 3.0)) 10
3.7. Example of communication in Dominion (Source DLR (CC-BY 3.0)) 11

5.1. Overview of the project 17
5.2. Long-term integration 18
5.3. TLConfigLoader sequence 19
5.4. SUMO's model of an intersection with its traffic lights 20
5.5. Basic structure of models 21
5.6. Flow of an adaptive signal control 23

6.1. TLConfigLoader's workflow 28
6.2. Control loop inside LightControl 29
6.3. Traffic light program 31
6.4. SPaT interface 32
6.5. TLGrabber's Logger class 33

A.1. Observer pattern 51

B.1. Model-View-Control pattern 53

D.1. SignalGroupController Diagram 57
D.2. TLConfigLoader methods 58
D.3. A traffic light program 59
D.4. LightControl class 60
D.5. SignalGroup class 61
D.6. TrafficLight class 62
D.7. TrafficLightsService Interface 62
D.8. NetworkController class 63
D.9. Intersection Class 63
D.10. IntersectionController class 64

List of Tables

7.1. Requirements and their solutions 35

Appendices

A Observer Design Pattern

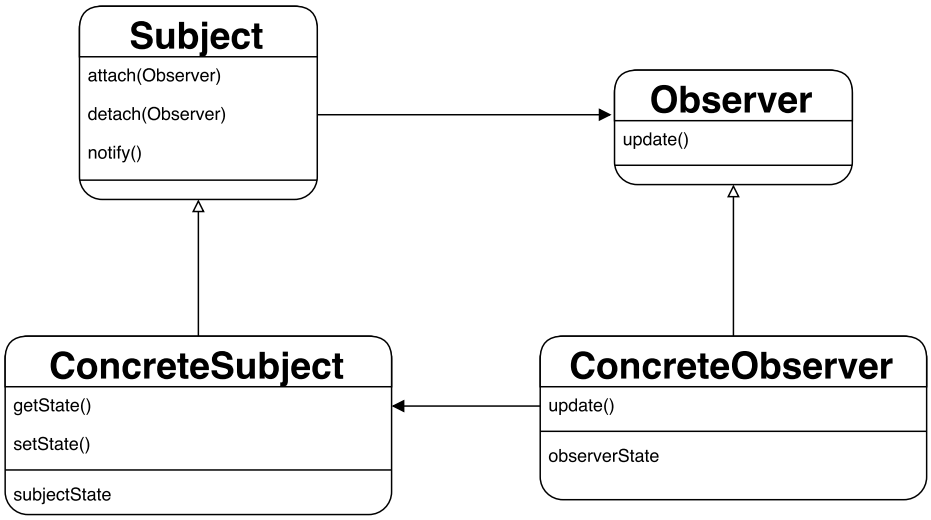


Figure A.1.: Observer pattern

The observer pattern is a behavioral pattern which allows to keep one object always up to date about another objects state. It works by defining abstract subjects and observer with the respective slot for each other, as can be seen in the diagram. After registering the observer at the subject they will be notified when anything interesting happens. This usually happens while executing a setter of some sort and in this method they invoke the notification for all registered observers.

B Model-View-Controller Pattern

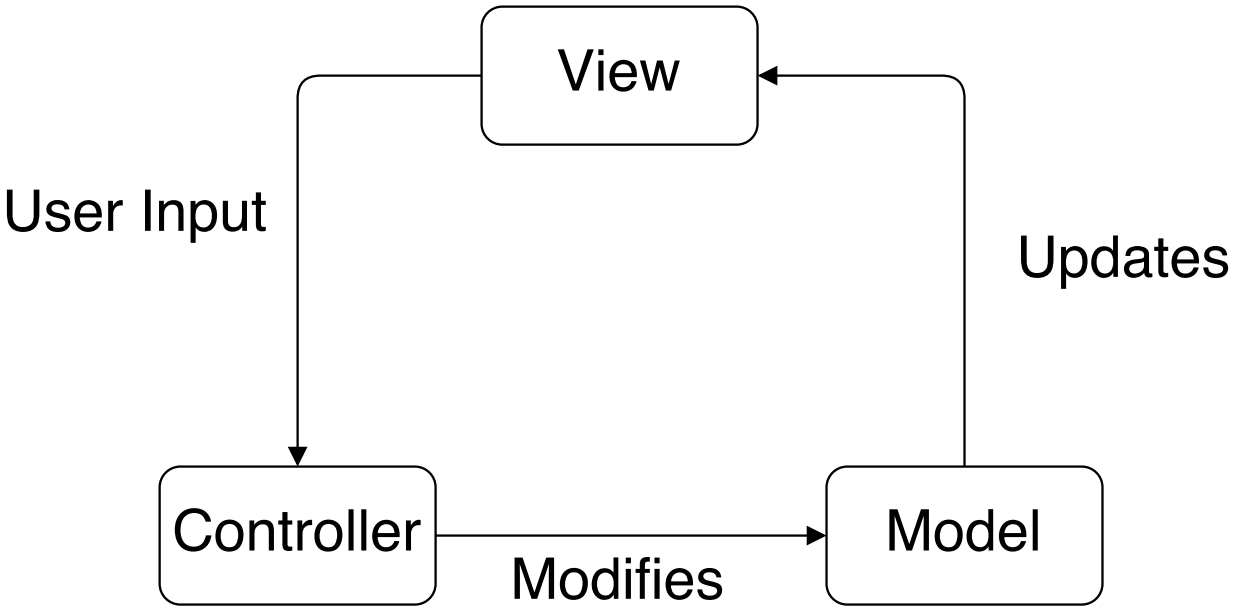
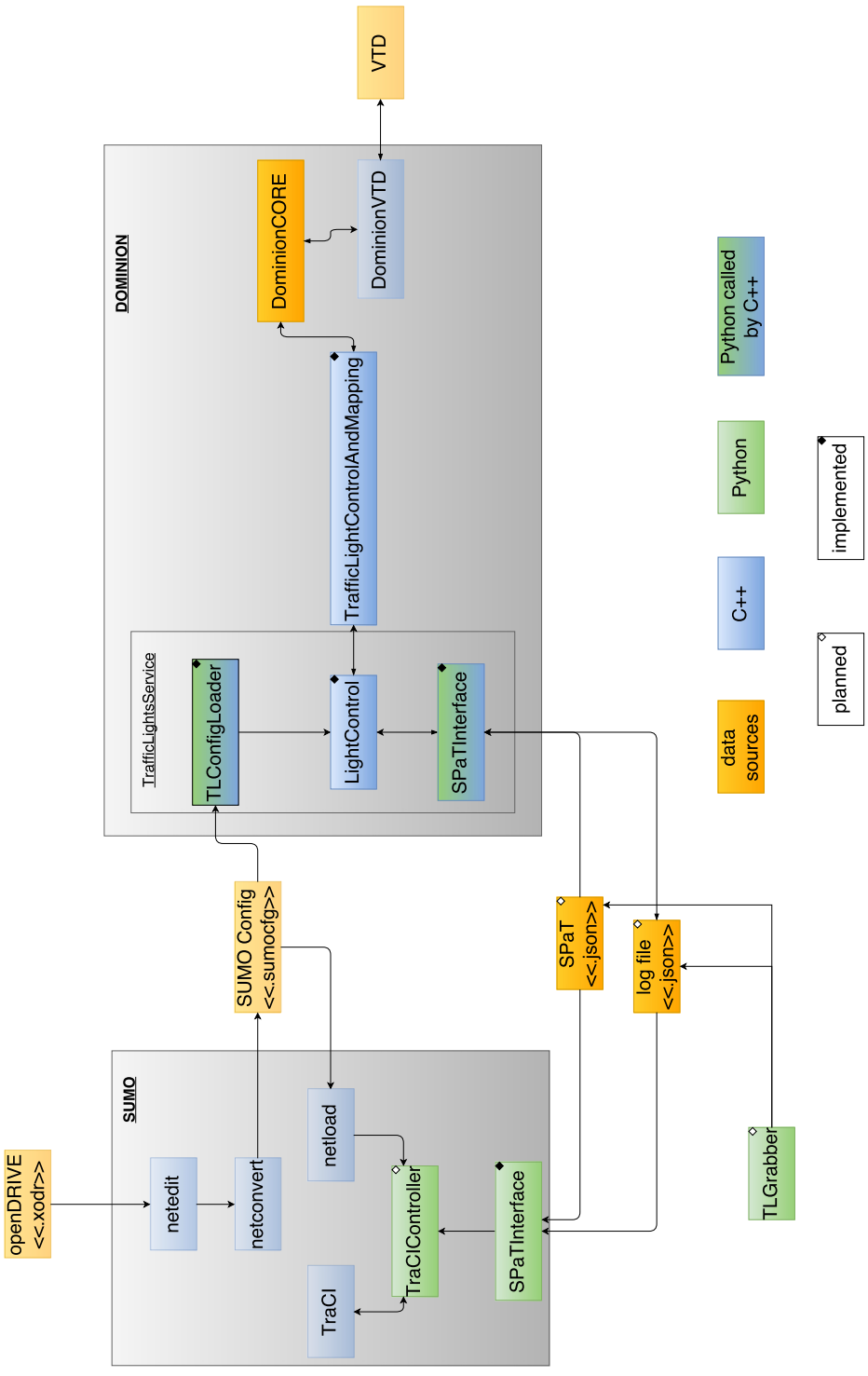


Figure B.1.: Model-View-Control pattern

Originally the model-view-controller pattern was designed for implementing user interfaces. By splitting the application into three parts, this modularity brought a gain in development efficiency and the possibility to store information in other ways that its is presented to the user. This pattern relies on the observer pattern as basis and has become quite popular for web applications and mobile applications for example. As the representation is separated from the model which holds the information and the controller which manipulates it, every part can be interchanged as long as the interfaces between the respective parts hold valid. In this thesis the view part was already given by Dominion and VTD®, so only the controller and model part had to be implemented.

C Design Overview



D Class Diagrams

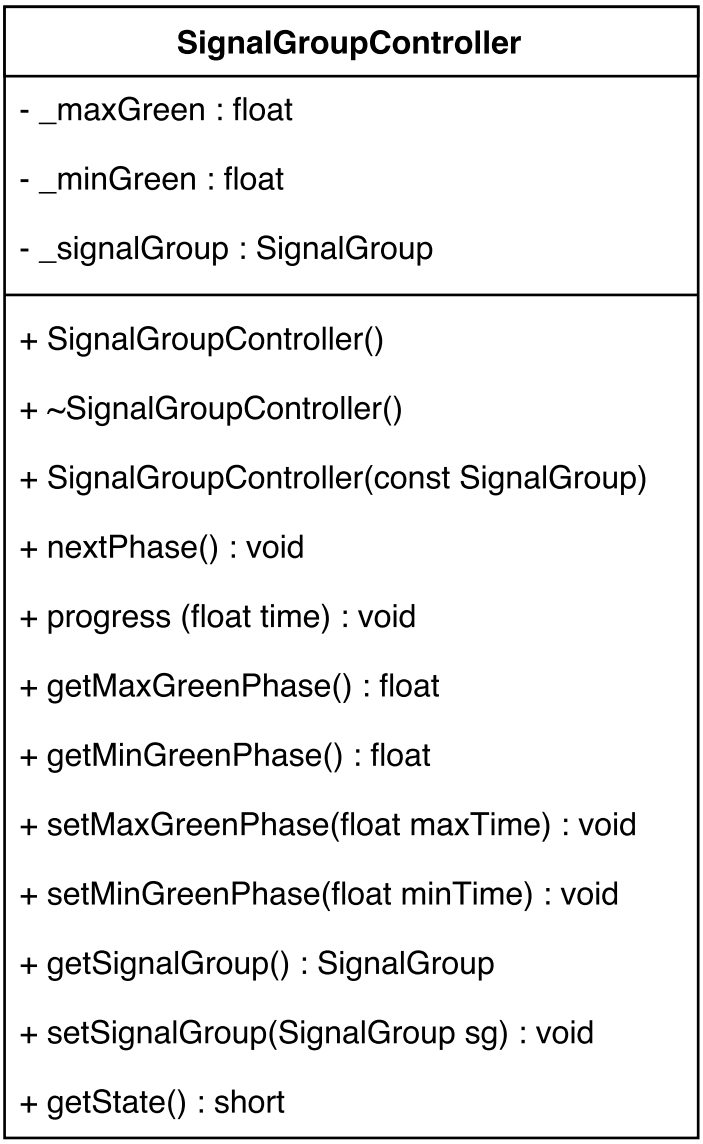


Figure D.1.: SignalGroupController Diagram

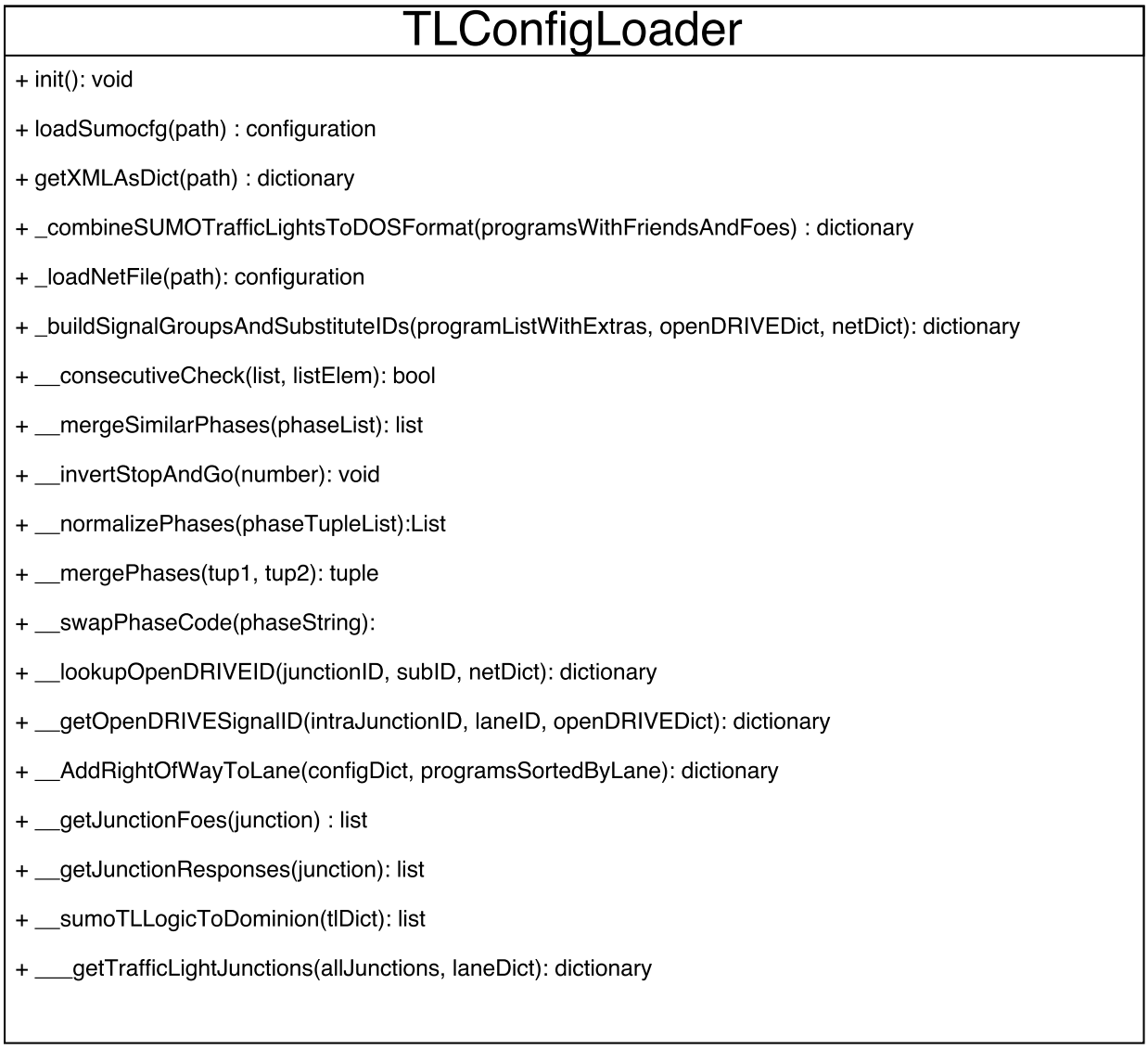


Figure D.2.: TLConfigLoader methods

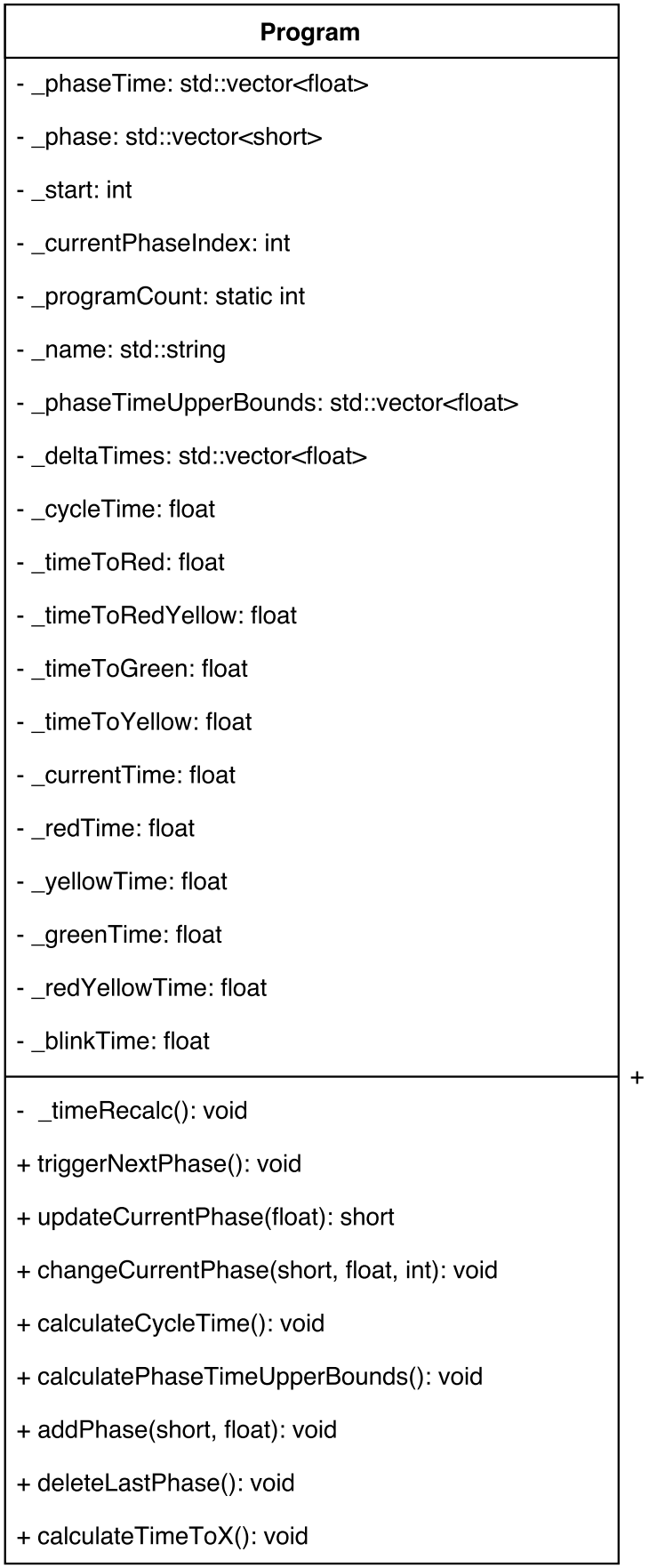


Figure D.3.: A traffic light program

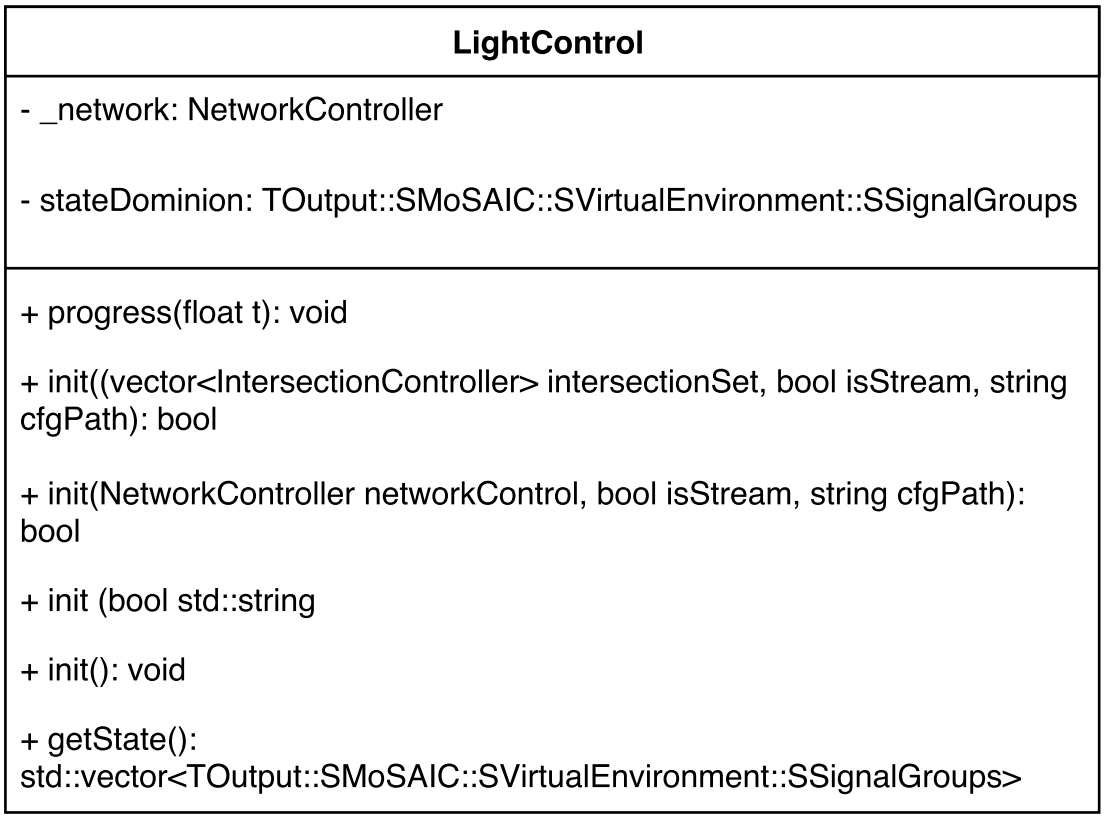


Figure D.4.: LightControl class

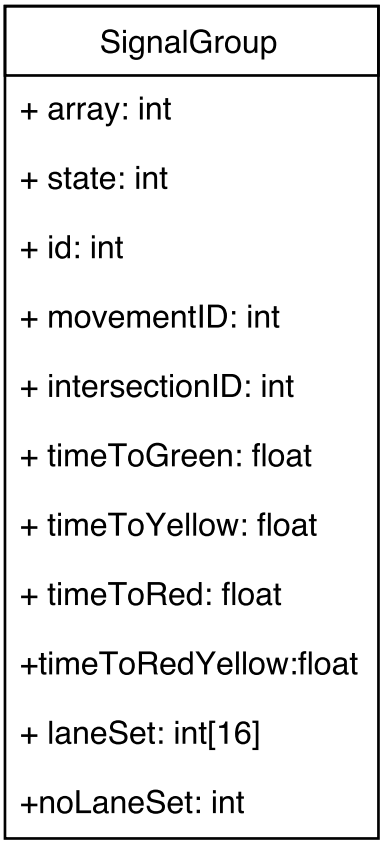


Figure D.5.: SignalGroup class

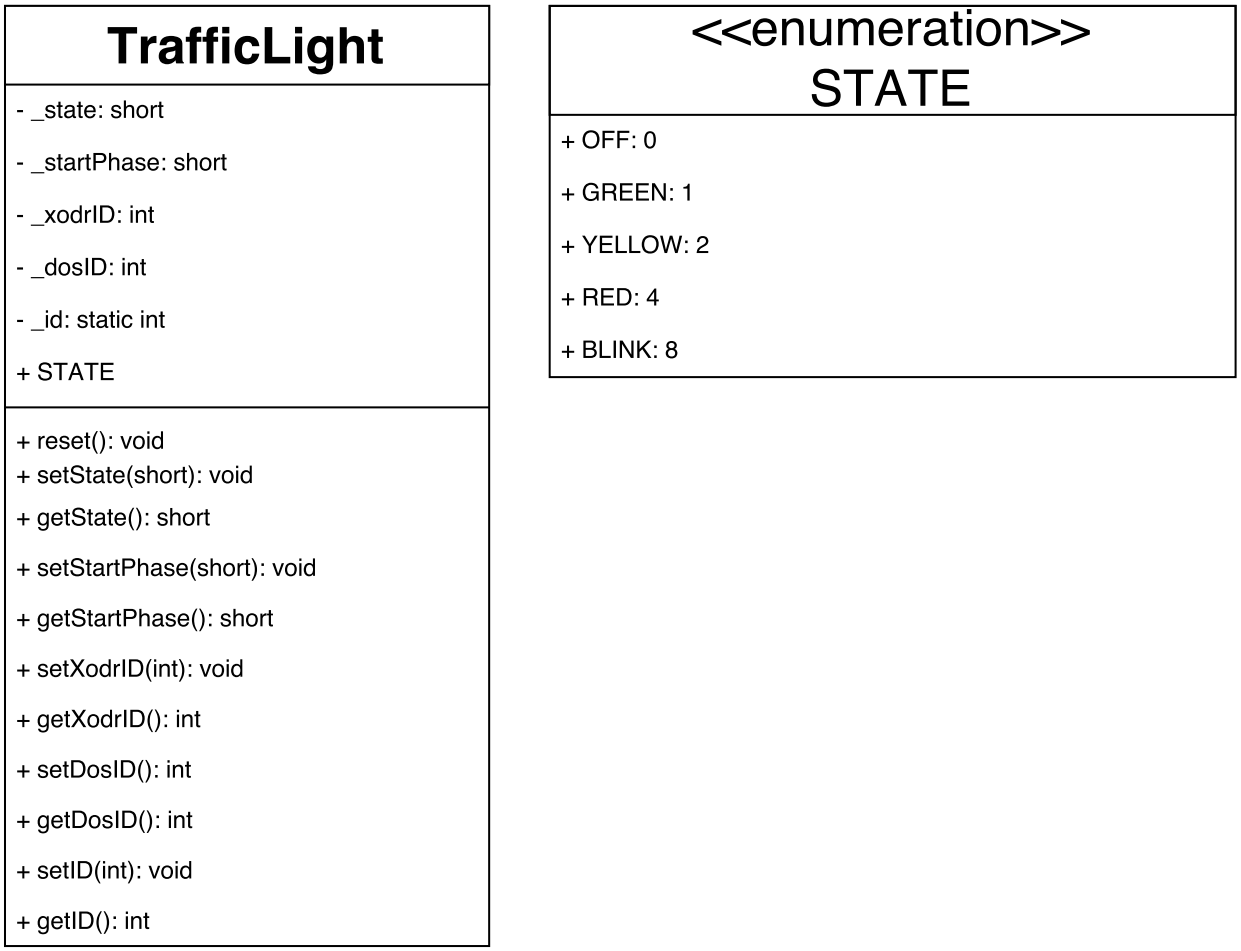


Figure D.6.: TrafficLight class

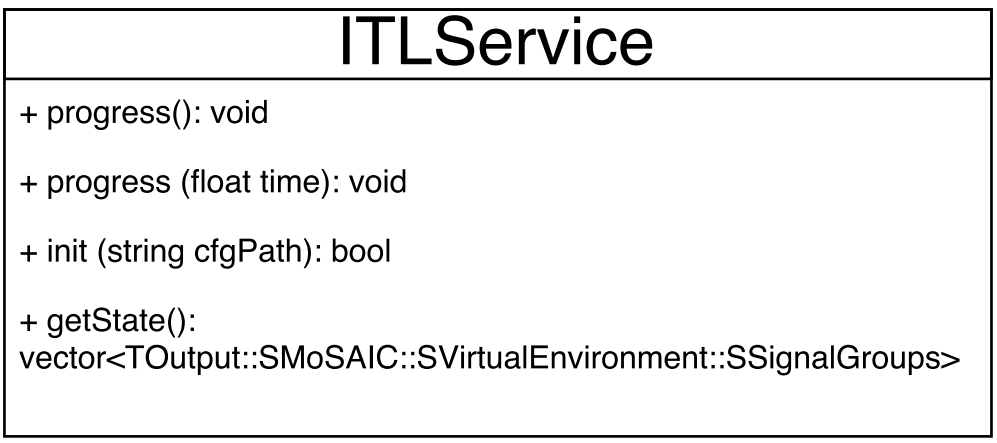


Figure D.7.: TrafficLightsService Interface

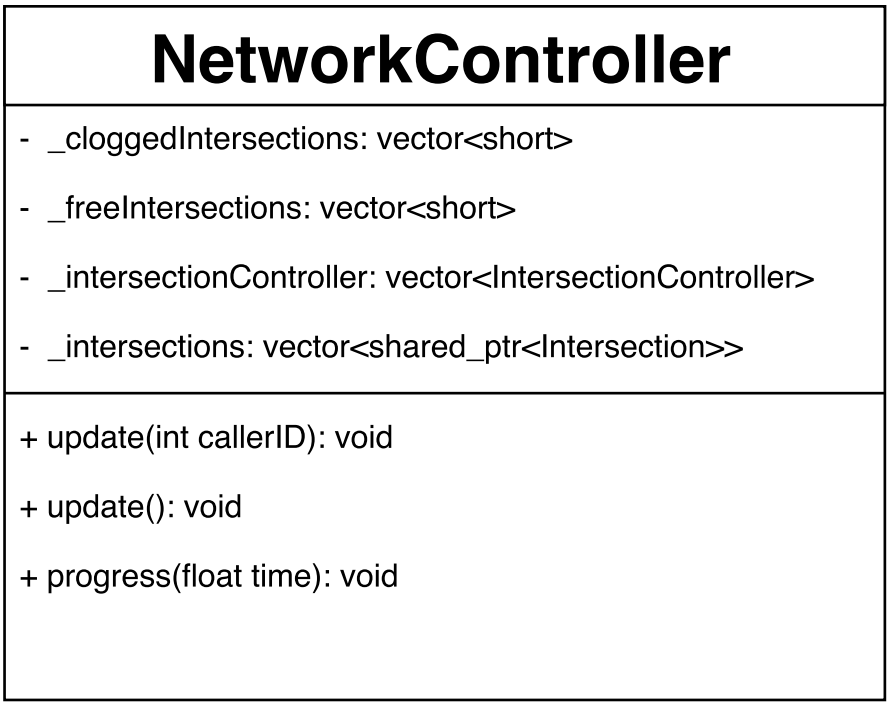


Figure D.8.: NetworkController class

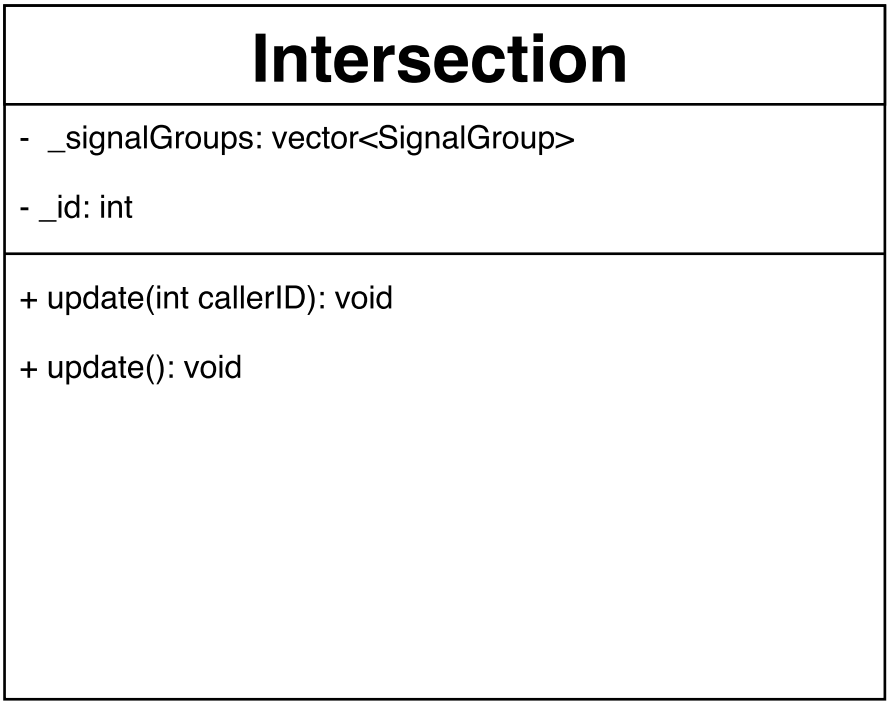


Figure D.9.: Intersection Class

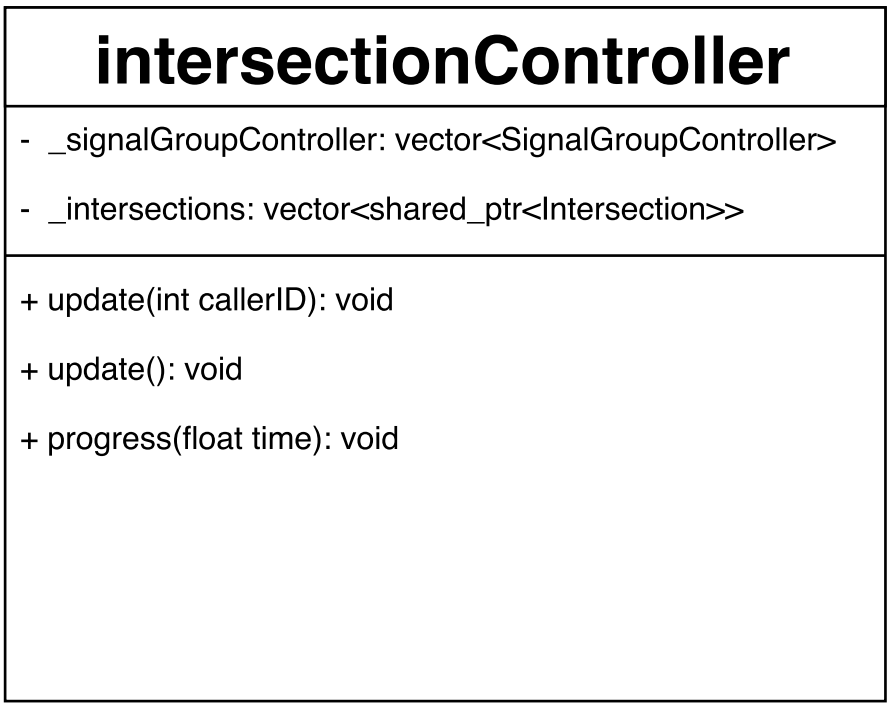


Figure D.10.: IntersectionController class

E Short instructions

E.0.1. Prepare map for SUMO

Convert for SUMO

```
netconvert --opendrive myOpenDriveNetwork.xodr -o mySUMOnetwork.
   net.xml --tls.guess-signals --output.original-names
```

Load into SUMO

Open the sumo-gui.exe and load the new net.xml file. Alternatively adjust an existing sumocfg file with random routes created with the randomRoutes script from the SUMO website.

Generating Traffic Lights

When the map has loaded, switch to traffic light mode and click on "'Generate traffic lights"' on the left hand side. Then save the newly generated configuration.

Loading programs into a DOS scenario

Optional: write a generator in Python according to the TLConfigLoader script that handles oddities on the given map.
Enter the path to the generated sumocfg file or net.xml file as value for the "'PROGRAM_PATH"' argument when starting TrafficLightControlAndMapping

Prof. Dr. Mladen Berekovic
Mühlenpfordtstr. 23
38106 Braunschweig