

# Consistency of Microscopic Traffic Domain Models

Institute for Transport Studies  
Department of Civil Engineering, Geo and Environmental Sciences  
Karlsruhe Institute of Technology

**Master's Thesis**

of

**Jelle Kübler**

Matriculation Number 1943108

**Reviewer:**

Prof. Dr.-Ing. Peter Vortisch

**Second Reviewer:**

Prof. Dr. rer. nat. Ralf H. Reussner

**Advisors:**

M.Sc. Lars Briem

Dr.-Ing. Erik Burger

Date of Submission: May 31, 2020  
Karlsruhe



# Declaration

I declare that I have developed and written the enclosed thesis without the assistance of others, and have not used sources or means without declaration in the text. External assistance includes paid services of third parties, e.g. Internet agencies, which check the written version for spelling and / or punctuation, grammar, formulation, logic, plausibility, sentence structure, sense or style. This work has not yet been submitted to another examination institution neither in the same nor in a similar way and has not yet been published. This thesis was carried out in accordance with the Rules for Safeguarding Good Scientific Practice at Karlsruhe Institute of Technology (KIT). I hereby grant the Institute for Transport Studies the right to use this thesis. This includes in particular the further use of the acquired knowledge and results in other works or publications by employees of the Institute for Transport Studies. I also grant the reference library of the Institute the right to use this thesis.

Karlsruhe, May 31, 2020

---

Jelle Kübler





# Aufgabenstellung für die Masterarbeit

von

**Jelle Kübler**

Matrikelnummer 1943108

mit dem Titel:

## **Konsistenz mikroskopischer Domänenmodelle in der Verkehrstechnik**

Englischer Titel: Consistency of microscopic traffic domain models

### **Problemstellung**

Verkehringenieure modellieren Verkehr auf verschiedenen Abstraktionsebenen, je nachdem, welcher Aspekt von Verkehr betrachtet wird. Es wird zwischen Mobilitätsforschung, Verkehrsplanung und Verkehrstechnik unterschieden. In der Verkehrsplanung sowie in der Verkehrstechnik werden meist Modelle verwendet, mit denen die Qualität des Verkehrsablaufs vorhergesagt werden soll. Bei der Verkehrsplanung werden üblicherweise große räumliche Ausschnitte des Verkehrsnetzes betrachtet, wie eine ganze Stadt oder Region, während in der Verkehrstechnik kleinere Ausschnitte wie Autobahnauffahrten oder einzelne Knotenpunkte untersucht werden. Aufgrund des starken Größenunterschieds des betrachteten Raums werden in der Verkehrsplanung und der Verkehrstechnik unterschiedliche Domänenmodelle mit unterschiedlichem Abstraktionsniveau angewandt. In der Verkehrsplanung wird eine abstraktere Repräsentation des Straßennetzes gewählt als in der Verkehrstechnik. Bei letzterer kommt es auf die genaue Modellierung der Geometrie der untersuchten Anlagen an. In der Verkehrsplanung hat sich die Repräsentation des Straßennetzes durch attributierte Graphen durchgesetzt. In der Verkehrstechnik hat sich dagegen keine einheitliche Repräsentation durchgesetzt. Die Domänenmodelle sind daher insgesamt sehr heterogen.

Aufgrund des unterschiedlichen Abstraktionsniveaus haben sich die Domänenmodelle in der Verkehrsplanung und der Verkehrstechnik unabhängig voneinander entwickelt. Eine Verknüpfung dieser Domänenmodelle würde die Effizienz des Nutzers verbessern, indem zum Beispiel die Verkehrsnachfrage eines Planungsmodells in ein verkehrstechnisches Modell übertragen werden kann, um dort die Widerstände von Knotenpunkten für das Planungsmodell zu bestimmen. Eine solche Verknüpfung über die Modellebenen hinweg ist jedoch nur möglich, wenn diese konsistent zueinander gehalten werden können. Heutzutage kann die Konsistenz der Instanzen nur durch einen manuellen, fehleranfälligen Prozess sichergestellt werden. Eine Automatisierung würde Fehler bei der Ausführung dieses Prozesses verringern. Die Heterogenität der verkehrstechnischen Domänenmodelle ist dabei eine der größten Herausforderungen. Um sicher zu entscheiden, welches Planungswerkzeug sich am besten für ein Projekt eignet, müssten Modelle in jedem davon manuell aufgebaut werden. Darum ist es wünschenswert, zwischen den verschiedenen Domänenmodellen wechseln zu können, um die jeweiligen Ergebnisse der Modellinstanzen vergleichen zu können. Die Konsistenzhaltung zwischen Verkehrsplanungs- und Verkehrstechnikmodellen würde sich dadurch deutlich vereinfachen.

## Aufgabenstellung

Im Hinblick auf die gewünschte Konsistenzhaltung von Verkehrsplanungs- und Verkehrstechnik-Domänenmodellen soll in dieser Arbeit das Problem der heterogenen verkehrstechnischen Domänenmodelle behandelt werden. Ziel dieser Arbeit ist es, einen effizienten Prozess zu entwickeln, mit dem Modellinstanzen verschiedener verkehrstechnischer Domänenmodelle konsistent gehalten werden können. Außerdem sollen die Simulationsergebnisse verschiedener Simulationsprogramme, angewendet auf konsistente Modellinstanzen, verglichen werden, um deren Einfluss auf das Ergebnis zu untersuchen.

Dazu sollen die folgenden zwei Simulationsprogramme bezüglich ihrer Domänenmodelle verglichen werden: das kommerziellen Simulationsprogramm Vissim (Verkehr In Städten - SimulationsModell) der PTV (Planung Transport Verkehr AG) und die Open Source Software SUMO (Simulation of Urban MObility) des deutschen Zentrums für Luft und Raumfahrt (DLR). Die gemeinsamen Konzepte der Programme sollen herausgearbeitet und in Form von zwei Metamodellen formal modelliert werden. Basierend auf den gefundenen Gemeinsamkeiten und Unterschieden soll eine Transformationssprache gewählt werden, mit der Instanzen des einen Metamodells in Instanzen des anderen überführt werden können. Des Weiteren soll untersucht werden, ob inkrementelle Änderungen zwischen den Modellinstanzen mithilfe von Konsistenzbedingungen propagiert werden können. Der Vitruvius-Ansatz (entwickelt am Lehrstuhl für Software-Entwurf und -Qualität (SDQ) am KIT) soll dafür evaluiert werden.

Zur Bewertung der entwickelten Konsistenzmechanismen, soll mit beiden Programmen ein exemplarisches Verkehrsnetz aufgebaut werden. Anhand dieses Netzes soll die Effektivitätssteigerung der automatischen gegenüber der manuellen Erstellung ermittelt werden. Basierend auf den konsistenten Modellen sollen anschließend die Simulationsergebnisse beider Programme exemplarisch verglichen werden.

Die Arbeit ist gebunden (DIN A4) in dreifacher Ausführung einzureichen und nach Möglichkeit ist die Formatvorlage des Instituts für Verkehrswesen zu verwenden.

Betreuer: Prof. Dr.-Ing. Peter Vortisch

Betreuende wissenschaftliche Mitarbeiter: M.Sc. Lars Briem

Karlsruhe, den 09.12.2019

Prof. Dr.-Ing. Peter Vortisch

ausgegeben: 01.01.2020

abzugeben: 01.06.2020





# Abstract

Traffic engineers model transportation on different levels of abstraction depending on the investigated aspects of transportation. In the fields of traffic planning and traffic engineering, models are used to predict the quality of transport infrastructure.

Due to the different levels of abstraction, domain models have developed independently in traffic planning and traffic engineering. In the latter case there are major differences between the various domain models. Deciding which modeling tool is most suitable for a project is critical but requires to build models with each of them manually before comparing them. Therefore, being able to switch between the different tools to use all of their respective features is desirable. However, this is only possible if the models can be kept consistent with each other while switching between different tools. Today, the consistency of instances of different domain models can only be ensured by a manual, error-prone process. Automating this process would reduce errors and increase the user's efficiency.

Model-Driven Development is an engineering paradigm that allows specifying systems on a higher abstraction level than object-oriented or component-based development. In Model-Driven Development, models are treated as primary artifacts that specify what functionality or structure a system should have. Within this field of science, different techniques have been developed to maintain the consistency of models.

In this thesis we apply these techniques to the domain of microscopic traffic simulation models to keep them consistent. We develop two metamodels which describe the structure of models used by the two microscopic traffic simulation programs PTV VISSIM and SUMO. We compare the different representations of similar concepts in both metamodels and develop relations between their elements. Based on the developed relations, we implement model transformations using the VITRUVIUS framework which propagate incremental changes between PTV VISSIM models and SUMO models to keep them consistent.

Finally, we evaluate the developed consistency mechanisms using a user study in which we determine their impact on the user's efficiency and effectiveness in creating consistent models. The evaluation shows a significant increase in efficiency and slightly more consistent models on average. Moreover, the consistency mechanisms received a higher rating in terms of system usability and overall user experience.



# Kurzfassung

Verkehringenieure modellieren Verkehr auf verschiedenen Abstraktionsebenen, je nachdem, welche Aspekte von Verkehr betrachtet werden. In der Verkehrstechnik werden meist Modelle verwendet, mit denen die Qualität des Verkehrsablaufs vorhergesagt werden soll. Die behandelten Konzepte werden in den verwendeten Domänenmodellen jedoch sehr unterschiedlich repräsentiert.

Um sicher zu entscheiden, welches Planungswerkzeug sich am besten für ein Projekt eignet, müssten Modelle in jedem davon manuell aufgebaut werden. Darum ist es wünschenswert, zwischen den verschiedenen Werkzeugen wechseln zu können, um die jeweiligen Vorteile zu verwenden. Dazu müsste jedoch sichergestellt werden, dass beim Wechsel der Werkzeuge die jeweiligen Modelle konsistent gehalten werden. Heutzutage kann die Konsistenz der Instanzen verschiedener verkehrstechnischer Domänenmodelle nur durch einen manuellen, fehleranfälligen Prozess sichergestellt werden. Eine Automatisierung würde Modellierungsfehler verringern und die Effizienz des Modellierers steigern.

Modellgetriebene Entwicklung ist ein Softwareentwicklungsparadigma, welches die Spezifikation von Systemen auf einer höheren Abstraktionsebene erlaubt als die objektorientierte oder komponentenbasierte Entwicklung. Bei der Modellgetriebenen Entwicklung werden Modelle als primäre Artefakte verwendet, um die Funktionalität oder die Struktur von Programmen zu spezifizieren. In diesem Forschungsfeld wurden einige Techniken zur konsistenten Modellierung entwickelt.

In dieser Arbeit werden diese Techniken auf verkehrstechnische Modelle angewendet, um diese konsistent zu halten. Zunächst werden zwei Metamodelle entwickelt, welche die Struktur von Modellen der Simulationsprogramme PTV VISSIM und SUMO beschreiben. Anschließend werden die unterschiedliche Modellierungen ähnlicher Konzepte in beiden Metamodellen verglichen und Beziehungen zwischen den einzelnen Elementen der Metamodelle herausgearbeitet. Basierend auf den herausgearbeiteten Beziehungen werden mithilfe des VITRUVIUS Ansatzes Transformationen implementiert, welche inkrementelle Änderungen zwischen PTV VISSIM Modellen und SUMO Modellen propagieren, um diese konsistent zu halten.

Zur Evaluation der entwickelten Konsistenzmechanismen wurde eine Benutzerstudie durchgeführt, in der die Effizienz des Nutzers bei der Erstellung konsistenter Modelle ermittelt wurde. Die Evaluation zeigt einen signifikanten Anstieg der Effizienz, sowie eine bessere Bewertung der Benutzerfreundlichkeit und generellen Nutzererfahrung.



# Contents

<b>Abstract</b>	<b>ix</b>
<b>Kurzfassung</b>	<b>xi</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xxi</b>
<b>List of Abbreviations</b>	<b>xxiii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Foundations</b>	<b>3</b>
2.1 Model-Driven Development . . . . .	3
2.1.1 General model theory . . . . .	3
2.1.2 Metamodeling . . . . .	5
2.1.3 Eclipse Modeling Framework . . . . .	7
2.1.4 Model transformations . . . . .	7
2.1.5 Consistent and view-based modeling . . . . .	9
2.1.6 Vitruvius . . . . .	11
2.2 Transport studies . . . . .	14
2.2.1 Transport planning . . . . .	14
2.2.2 Traffic engineering . . . . .	15
2.2.3 Coupling macroscopic and microscopic models . . . . .	15
2.2.4 PTV Vissim . . . . .	16
2.2.5 SUMO - Simulation of Urban Mobility . . . . .	17
<b>3 Metamodels</b>	<b>19</b>
3.1 Modeling Transport Infrastructure . . . . .	22
3.1.1 Roads and Intersections . . . . .	22
3.1.2 Speed limits . . . . .	27
3.1.3 Stop Lines at Intersections . . . . .	30
3.1.4 Public Transport Stops and Parking Lots . . . . .	33

3.2	Modeling Traffic Control . . . . .	37
3.2.1	Right of Way at Intersections . . . . .	37
3.2.2	Signal Programs . . . . .	42
3.3	Modeling Traffic . . . . .	50
3.3.1	Vehicle Properties . . . . .	50
3.3.2	Driving Behavior . . . . .	54
3.3.3	Traffic Demand . . . . .	61
3.4	Modeling Output Measurements . . . . .	73
3.4.1	Detectors in SUMO . . . . .	73
3.4.2	Measurements in PTV Vissim . . . . .	76
<b>4</b>	<b>Model Comparison</b>	<b>79</b>
4.1	Comparing Roads and Intersection . . . . .	79
4.2	Comparing Speed limits . . . . .	83
4.3	Comparing Stop Lines . . . . .	85
4.4	Comparing Stop Infrastructure . . . . .	86
4.5	Comparing Right of Way at Intersections . . . . .	88
4.6	Comparing Traffic Light Programs . . . . .	91
4.7	Comparing Vehicle Properties . . . . .	94
4.8	Comparing Driving Behavior . . . . .	98
4.9	Comparing Traffic Demand . . . . .	98
4.10	Comparing Measurements . . . . .	102
<b>5</b>	<b>Implementation</b>	<b>105</b>
5.1	Text to Model Transformations . . . . .	106
5.2	Model to Text Transformations . . . . .	109
5.3	Reactions . . . . .	110
<b>6</b>	<b>Evaluation</b>	<b>113</b>
6.1	Research Question and Hypotheses . . . . .	113
6.2	Methods . . . . .	114
6.3	Sample . . . . .	115
6.4	Procedure . . . . .	116
6.5	Results . . . . .	117
<b>7</b>	<b>Conclusion</b>	<b>121</b>
7.1	Results . . . . .	121
7.2	Outlook . . . . .	121
	Bibliography . . . . .	123
<b>A</b>	<b>Appendix</b>	<b>129</b>

# List of Figures

2.1	The seven bridges of Königsberg . . . . .	4
2.2	Four modeling layers and example instances . . . . .	6
2.3	An overview on model transformation . . . . .	8
2.4	Synthetic and projective views . . . . .	10
3.1	Four modeling layers applied to PTV VISSIM and SUMO models . . . . .	19
3.2	Metamodel notation . . . . .	20
3.3	Example Connections in SUMO . . . . .	23
3.4	Edge-shape example in SUMO . . . . .	23
3.5	SUMO Metamodel: Roads and Intersections . . . . .	24
3.6	Example Links in PTV VISSIM . . . . .	25
3.7	Example Connectors in PTV VISSIM . . . . .	26
3.8	PTV VISSIM Metamodel: Roads and Intersections . . . . .	27
3.9	SUMO Metamodel: Speed Limits . . . . .	28
3.10	Speed limit example in PTV VISSIM . . . . .	29
3.11	PTV VISSIM Metamodel: Speed Limits . . . . .	30
3.12	SUMO Metamodel: Stop Lines . . . . .	31
3.13	Stop line example in PTV VISSIM . . . . .	32
3.14	PTV VISSIM Metamodel: Stop Lines . . . . .	32
3.15	SUMO Metamodel: Public Transport Stops and Parking Lots . . . . .	34
3.16	Parking Lot example in SUMO . . . . .	35
3.17	Bus Stop example in SUMO . . . . .	35
3.18	PTV VISSIM Metamodel: Public Transport Stops and Parking Lots . . . . .	36
3.19	Example Parking Lots and Public Transport Stops in PTV VISSIM . . . . .	36

3.20 SUMO Metamodel: Right of Way at Intersections . . . . .	39
3.21 PTV VISSIM Metamodel: Right of Way at Intersections . . . . .	41
3.22 Example Conflict Areas in PTV VISSIM . . . . .	41
3.23 Conflict Areas at branching Links . . . . .	41
3.24 Priority Rule example in PTV VISSIM . . . . .	41
3.25 SUMO Metamodel: Signal Programs . . . . .	44
3.26 Signal program example in SUMO . . . . .	44
3.27 PTV VISSIM Metamodel: Signal Controllers . . . . .	46
3.28 Signal Programs example in PTV VISSIM . . . . .	47
3.29 Stage Program example in PTV VISSIM . . . . .	47
3.30 VISSIG Metamodel: Signal Programs . . . . .	49
3.31 SUMO Metamodel: Vehicle Properties . . . . .	51
3.32 PTV VISSIM Metamodel: Vehicle Properties . . . . .	53
3.33 Example Acceleration Function in PTV VISSIM . . . . .	53
3.34 Interaction states of the Wiedemann model . . . . .	55
3.35 SUMO Metamodel: Driving Behavior . . . . .	56
3.36 SUMO Metamodel: Car Following Models . . . . .	57
3.37 PTV VISSIM Metamodel: Driving Behavior . . . . .	59
3.38 PTV VISSIM Metamodel: Link dependent Driving Behavior . . . . .	61
3.39 SUMO Metamodel: Traffic Demand . . . . .	63
3.40 SUMO Metamodel: Vehicle Stops . . . . .	64
3.41 Rerouting example in SUMO . . . . .	64
3.42 SUMO Metamodel: Rerouters . . . . .	65
3.43 Vehicle Input example in PTV VISSIM . . . . .	66
3.44 PTV VISSIM Metamodel: Traffic Demand . . . . .	67
3.45 Vehicles without Routes in PTV VISSIM . . . . .	67
3.46 PTV VISSIM Metamodel: Vehicle Routes . . . . .	69
3.47 Vehicle Routes to Parking Lots in PTV VISSIM . . . . .	69
3.48 Vehicle Route example in PTV VISSIM . . . . .	70



3.49	PTV VISSIM Metamodel: Public Transport Lines . . . . .	71
3.50	Public Transport Line example in PTV VISSIM . . . . .	72
3.51	Partial Public Transport Line example in PTV VISSIM . . . . .	72
3.52	SUMO Metamodel: Detectors . . . . .	75
3.53	Lane Detector example in SUMO . . . . .	75
3.54	Multilane Detector example in SUMO . . . . .	75
3.55	Entry-Exit Detector example in SUMO . . . . .	75
3.56	Example Measurements in PTV VISSIM . . . . .	76
3.57	PTV VISSIM Metamodel: Measurements . . . . .	78
4.1	Transforming PTV VISSIM Links to SUMO Edges . . . . .	80
4.2	Possible PTV VISSIM representations of a SUMO Edge sequence . . . . .	80
4.3	Example violation of the PUTGET rule . . . . .	81
4.4	Correspondence of SUMO Connections and PTV VISSIM Connectors . . . . .	81
4.5	Consistency of Stop Lines in PTV VISSIM and SUMO. . . . .	86
4.6	Consistency of PTV VISSIM public transport stops and SUMO bus stops . . . . .	87
4.7	Correspondence of vehicle acceleration in PTV VISSIM and SUMO . . . . .	95
4.8	Correspondence of an acceleration distribution in PTV VISSIM and SUMO . . . . .	96
4.9	Correspondence of PTV VISSIM Routing Decisions and SUMO Rerouters . . . . .	100
4.10	Correspondence of Travel Time Measurements in PTV VISSIM and SUMO . . . . .	103
5.1	Implemented model transformations . . . . .	105
5.2	Model to Model transformations with VITRUVIUS . . . . .	110
6.1	Evaluation of Efficiency - Task completion times . . . . .	117
6.2	Evaluation of Efficiency - User Experience . . . . .	118
6.3	Evaluation of Effectiveness - Model Consistency . . . . .	119
6.4	Evaluation of System Usability and User Experience . . . . .	120
A.1	Description of SUMO's editor modes . . . . .	129
A.2	Description of SUMO's infrastructure objects . . . . .	130

A.3	Description of SUMO's demand objects . . . . .	131
A.4	Description of SUMO's evaluation objects . . . . .	132
A.5	Description of SUMO's configuration file . . . . .	133
A.6	Description of PTV VISSIM's editor modes . . . . .	133
A.7	Description of PTV VISSIM's infrastructure objects . . . . .	134
A.8	Description of PTV VISSIM's demand objects . . . . .	135
A.9	Description of PTV VISSIM's evaluation objects . . . . .	136
A.10	Description of PTV VISSIM's evaluation configuration . . . . .	137
A.11	Modeling task . . . . .	138
A.12	User Experience Questionnaire . . . . .	139
A.13	System Usability Scale . . . . .	140

# Listings

2.1	An exemplary VITRUVIUS reaction . . . . .	11
2.2	An exemplary VITRUVIUS routine . . . . .	12
5.1	SAX callback method startElement . . . . .	107
5.2	An exemplary Job to resolve cross references . . . . .	107
5.3	SAX callback method endElement . . . . .	108
5.4	Xtend-template for the XML representation of a Link . . . . .	109



# List of Tables

4.1	Example signal program matrix in SUMO . . . . .	92
4.2	Vehicle Type parameter combinations and their probabilities . . . . .	97



# List of Abbreviations

<b>CMOF</b>	Complete MOF
<b>DOM</b>	Document Object Model
<b>EMF</b>	Eclipse Modeling Framework
<b>EMOF</b>	Essential MOF
<b>HBS</b>	Handbuch für die Bemessung von Straßenverkehrsanlagen
<b>M2M</b>	Model-To-Model
<b>M2T</b>	Model-To-Text
<b>MDD</b>	Model-Driven Development
<b>MOF</b>	Meta-Object Facility
<b>OCL</b>	Object Constraint Language
<b>OSM</b>	Orthographic Software Modeling
<b>PTV</b>	PTV Planung Transport Verkehr AG
<b>PTV Vissim</b>	Verkehr In Städten - SimulationsModell
<b>QVT</b>	Query, View, Transformation
<b>SAX</b>	Simple API for XML
<b>SUM</b>	Single Underlying Model
<b>SUMO</b>	Simulation of Urban MObility
<b>SUS</b>	System Usability Scale
<b>T2M</b>	Text-To-Model
<b>UEQ</b>	User Experience Questionnaire
<b>UML</b>	Unified Modeling Language
<b>VisSig</b>	Signal Program Model of PTV VISSIM
<b>Vitruvius</b>	View-centric engineering Using a Virtual Underlying Single model
<b>VSUM</b>	Virtual Single Underlying Model
<b>WAUT</b>	Wochenschaltautomatik (~ weekly switch automatism)
<b>XMI</b>	XML Metadata Interchange
<b>XML</b>	Extensible Markup Language
<b>XSD</b>	XML Schema Definition





# 1 Introduction

Transportation in general has different levels of abstraction targeting different aspects of traffic, namely mobility behavior research, transport planning, and traffic engineering. In the fields of traffic planning and traffic engineering, models are used to predict the quality of transport infrastructure. In traffic planning, transportation is usually investigated at a large spatial scope such as an entire city or region, while in traffic engineering smaller sections like single motorway accesses or individual intersections are examined. Due to that difference in scale, different domain models at different levels of abstraction are used in transport planning and traffic engineering.

Transport planning models usually share a more abstract representation of the road network than traffic engineering models. In the latter case, it is important to model the exact geometry of the examined facilities. In transport planning, the representation of road networks by attributed graphs is commonly accepted. In traffic engineering, however, no uniform representation has been established. Traffic engineering domain models are therefore very heterogeneous.

Due to the different levels of abstraction, domain models have developed independently in traffic planning and traffic engineering. Linking these domain models would improve the efficiency of the user, e.g. by transferring the traffic demand of a planning model into a traffic engineering model which in turn can be used to determine the impedance of nodes for the planning model. However, linking models of different abstraction levels is only possible if they are consistent. Today, the consistency of models can only be ensured by a manual, error-prone process. Automating this process would reduce errors and increase the user's efficiency. The heterogeneity of the traffic engineering domain models poses one of the biggest challenges. Deciding which modeling tool is most suitable for a project, requires to build models with each of them manually before comparing them. Therefore, being able to switch between the different domain models with less manual effort is a desirable skill. This would allow traffic engineers to switch between simulation programs of different vendors to use their unique features. Moreover, it would significantly simplify keeping transport planning and traffic engineering models consistent.

The goal of this thesis is to develop consistency mechanisms to keep models of two exemplary traffic simulation programs consistent: the widely used commercial simulation program PTV VISSIM<sup>1</sup> [1] (version 11.00 - 12) and the open-source software SUMO<sup>2</sup> (Simulation of Urban Mobility) [2] (version 1.3) of the German Aerospace Center (DLR).

To approach this goal we apply techniques from Model-Driven Development to keep these models consistent. At first, we develop the metamodels of both simulation programs which describe the structure of the static models that are used as input for the traffic simulation. Both metamodels are reverse engineered from exemplary models and the documentation of the respective simulation program. Both metamodels cover the most important aspects of their respective simulation program including network infrastructure, stop infrastructure, traffic rules, vehicle properties, driving behavior and traffic demand. They are described as instances of the *Ecore*-meta-metamodel which is the core of the Eclipse Modeling Framework. Based on the two metamodels, we compare their representations of similar concepts and develop relations between their model elements. Finally, we show, how the VITRUVIUS<sup>3</sup> approach [3] can be used to implement consistency mechanisms based on the developed relations which propagate incremental changes between the models. Being able to propagate incremental changes is important since not all model elements or attributes have a correspondence in the other model. Values specified by the user would be lost if the entire network were to be regenerated upon every change. Propagating incremental changes allows to only modify corresponding elements while keeping the custom values of unrelated model elements.

Chapter 2 gives a short introduction to Model-Driven Development in section 2.1 including general model theory in section 2.1.1 and fundamental techniques and concepts behind model driven development like metamodeling, model transformations and consistent and view-based development in sections 2.1.2, 2.1.4 and 2.1.5. Furthermore, the used tools VITRUVIUS and the Eclipse Modeling Framework are introduced in sections 2.1.3 and 2.1.6. The chapter also contains an introduction to transport studies in section 2.2 and the two simulation programs PTV VISSIM and SUMO in sections 2.2.4 and 2.2.5.

In Chapter 3 we present the developed metamodels for PTV VISSIM and SUMO divided into four aspects: infrastructure, traffic control, traffic and evaluation in sections 3.1 to 3.4.

In chapter 4 we develop the relations between the two developed metamodels in sections 4.1 to 4.10. The implementation of the transformations in VITRUVIUS is described in chapter 5.

Finally, in chapter 6 we present a study that was conducted to estimate the impact of the developed consistency mechanisms on the modeling process. section 6.5 describes the results of that study. Chapter 7 summarizes this thesis and gives an outlook on potential future work.

---

<sup>1</sup>Verkehr In Städten - SimulationsModell

<sup>2</sup>Simulation of Urban MObility

<sup>3</sup>View-centric engineering Using a Virtual Underlying Single model

## 2 Foundations

In this chapter we will give a short introduction to the theoretical foundations of Model Driven Development in section 2.1 as well as an introduction to transportation studies and the two microscopic traffic simulation programs PTV VISSIM and SUMO in section 2.2.

### 2.1 Model-Driven Development

Model-Driven Development is a programming paradigm that seeks to further raise the level of abstraction on which programs can be specified. That level of abstraction on which software development takes place has risen from procedural programming over object-oriented programming to component-based development [4]. In MDD<sup>1</sup>, models are treated as primary artifacts. They are used to specify what functionality or structure a system should have. Those models can then be automatically transformed into code for a specific platform which reduces (platform) complexity during development.

Since models are a core aspect of MDD, we will first establish a common understanding of the term ‘model’ in section 2.1.1 and have a look at metamodeling in section 2.1.2, which allows us to specify the structure of models. Aside from models, model transformations are a second essential part of MDD. With model transformations, existing models can be transformed into new models that are more suitable for specific use cases. Model transformations are treated in section 2.1.4. Finally section 2.1.5 gives an overview on consistent and view-based modeling, which aims at keeping multiple models with redundant information consistent.

#### 2.1.1 General model theory

The understanding of the terms ‘model’ and ‘modeling’ varies across different scientific fields. In formal logic, a model is a satisfying assignment of variables in a logical formula. However, in the field of software engineering, (domain-)models are thought of as abstractions of real-world or imaginary objects. To unite these different concepts of ‘models’ and ‘modeling’ Stachowiak [5] proposes three features that all models have in common.

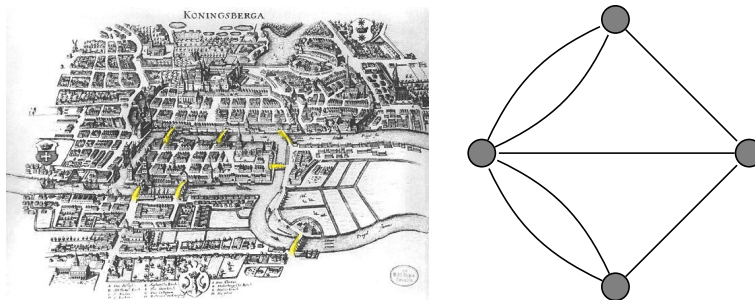
According to his general model theory, each model has a representation feature, a reduction

---

<sup>1</sup>Model-Driven Development

feature and a pragmatic feature [5, sec. 2.1.1]. The representation feature states that a model is always a model of something. These ‘things’ represented by a model are called ‘originals’ which can be either natural or artificial. Moreover, such an original can be a model itself. Usually models do not reflect all attributes of the originals they represent. They reduce the level of detail by only providing those attributes that seem relevant to the creators or users of the model. This second feature already gives the model a pragmatic character. More specifically, a model’s pragmatic feature consists of three aspects: A model is a model for someone (a natural or artificial user), it performs its substitution function during a certain period of time and under restriction to conceptual or actual operations.

For example, consider the Prussian city Königsberg where Leonard Euler showed that the Königsberg bridge problem has no solution. The problem was to find a path (today called ‘Euler walk’) through the city that crosses each bridge exactly once (see the highlighted bridges in figure 2.1). With his theorem, Euler laid the ground for what is now known as graph theory. To prove his theorem, he abstracted from the actual city and used a graph to argue about paths [6]. The graph’s nodes represent the different parts of the city that are divided by the river and its edges represent the bridges between them as shown in figure 2.1 on the right. In this case, the graph is a model of the city and its bridges: It has the representation feature as it represents the city Königsberg, which is the original in this example. Also the graph only represents the topology of the city, while leaving out its exact geometry or the length of the individual bridges. Therefore, it has the reduction feature. Finally, the graph also has the pragmatic feature, since it is a model for Euler or anyone trying to solve the Königsberg bridge problem. Also it models Königsberg around 1740 and is used to formally argue about or calculate paths and cycles.



**Figure 2.1:** A map of Königsberg with highlighted bridges (yellow) on the left (based on [7]) and an abstraction of Königsberg as a graph on the right.

Models can be distinguished into prescriptive and descriptive models [8, p. 181]. Prescriptive models exist before the original they represent and serve as a blueprint. In this case, the original is said to instantiate the model. If the original exists before the model, the latter is called descriptive. In the previous example, the graph representing Königsberg is a descriptive model since the city existed before Euler decided to take a walk across its bridges. The process of creating a model from an existing system is called reverse engineering.

### 2.1.2 Metamodeling

A metamodel is a model describing other models. The prefix 'meta' is a Greek prefix meaning 'above' or 'beyond'. As the prefix indicates, a metamodel has a higher level of abstraction than the models it describes because it is used to specify their structure. Thus, every model is an instance of a metamodel [9, p. 109].

In order to specify a metamodel, the following aspects need to be covered:

- The abstract syntax describes the potential structure of models. It defines the constructs of a modeling language and the relationships between those constructs without defining a concrete representation [9, p. 108ff]. This abstract representation is the basis for the automated processing of models.
- A concrete syntax describes how the model elements specified in an abstract syntax are represented. There can be multiple concrete syntaxes for one abstract syntax. For example, the constructs of a metamodel can be represented either textually, graphically or tree-based. The quality of the representation highly influences e.g. the readability and usability of a modeling language [9, p. 108ff].
- A model's context conditions describe additional modeling rules that cannot be expressed by the abstract syntax [10]. These rules constrain the set of model instances and specify which of them are well-defined. A common language to specify such constraints is OCL<sup>1</sup> [11]. In the context of UML<sup>2</sup>, the term 'static semantics' is used, which Harel and Rumpe [10, p. 65, 68] argue to be misleading since it merely constrains the syntax but does not contribute to the definition of semantics.
- The dynamic semantics of a metamodel defines the meaning of its constructs and their relationships [9]. These semantics are often given in the form of an informal natural language text. A more formal approach is to create a mapping of the metamodel to a language with defined semantics (e.g. Petri-networks). With a formal specification of semantics, desired properties can be proven valid.

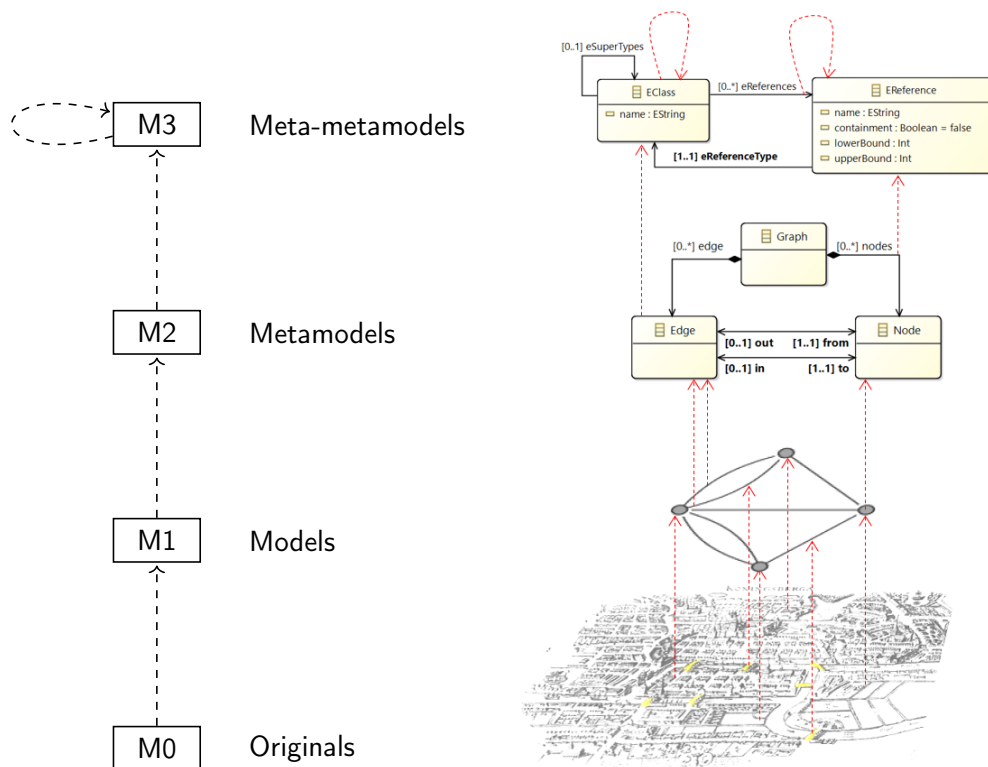
Considering metamodels describe (or 'model') the structure of models, they are models themselves. Thus, for a given metamodel, there exists another metamodel which in turn describes the structure of the given metamodel. That second metamodel is called meta-metamodel. This stack of increasingly abstract modeling layers can be extended until a self-describing metamodel is reached. A self-describing model is its own metamodel. The Meta-Object Facility (MOF) standard is an example of such a self-describing metamodel [12].

---

<sup>1</sup>Object Constraint Language

<sup>2</sup>Unified Modeling Language

The UML specification [13] suggests four modeling layers  $M0$ - $M3$ , where  $M3$  is the most abstract and  $M0$  the most concrete layer. Figure 2.2 shows the four modeling layers with instance-of relationships between them on the left. The right side of figure 2.2 shows exemplary instances of each modeling layer and how the elements of a layer instantiate elements of the next one. An example of a self-describing meta-metamodel at layer  $M3$  is *Ecore*. The model element *EClass* describing software classes is itself a class and the reference *eReferences* instantiates the class *EReference*. Layer  $M2$  contains metamodels. Continuing the example from section 2.1.1 this could be a graph-metamodel describing the elements like nodes and edges. Concrete instances of the metamodel are contained in layer  $M1$ , for example a graph modeling the topology of Königsberg. Finally, level  $M0$  consists of originals which is the city Königsberg in our example.



**Figure 2.2:** The four modeling layers  $M0$ - $M3$  on the left and an instance of each layer on the right with exemplary instance-of relationships (dashed arrows).

### 2.1.3 Eclipse Modeling Framework

The Eclipse Modeling Framework is an Eclipse-based framework that integrates models into the software development process. It provides the meta-metamodel *Ecore*, which is compatible with Essential MOF<sup>1</sup>. EMOF<sup>2</sup> is a self-describing meta-metamodel and a subset of Meta-Object Facility. While CMOF<sup>3</sup> provides the full power of MOF, EMOF only provides concepts to support object-oriented languages [12].

EMF allows to create *Ecore*-based metamodels either by hand with an editor or by importing UML files, XML<sup>4</sup> schemes or annotated Java-interface files. EMF also includes tools for generating Java-code and editors to create instances of an *Ecore*-metamodel. For persisting models, the XMI<sup>5</sup> format is used.

### 2.1.4 Model transformations

Model transformations are ‘the heart and soul’ [14] of model driven development as they are used to perform automated processes on models. A model transformation is the process of generating a target model from a given source model. The process of producing a target model is specified by a set of transformation rules [15]. Such a transformation rule describes how one or more constructs of the source metamodel are transformed into one or more constructs of the target metamodel.

Figure 2.3 shows an overview of the most important components in a model transformation. The actual transformation is performed by a transformation engine, which receives a source model and produces a target model. The engine’s second input is a set of transformation rules (written in a specific transformation language) which describe how source model elements are transformed into target model elements. Therefore, they reference the metamodels of the source and target model.

Model transformations can be classified in four dimensions [17]:

- For one, the technical space of the source- and target-model can either differ or be the same. A technical space is a model management framework and is determined by the meta-metamodel [17, p. 130]. The overview in figure 2.3 shows a scenario, where the source and target metamodel have the same meta-metamodel, thus the transformation stays in the same technical space.
- Secondly, the source and target model can have the same or different metamodels [17, p. 131f]. If their metamodels are different the transformation is called exogenous.

---

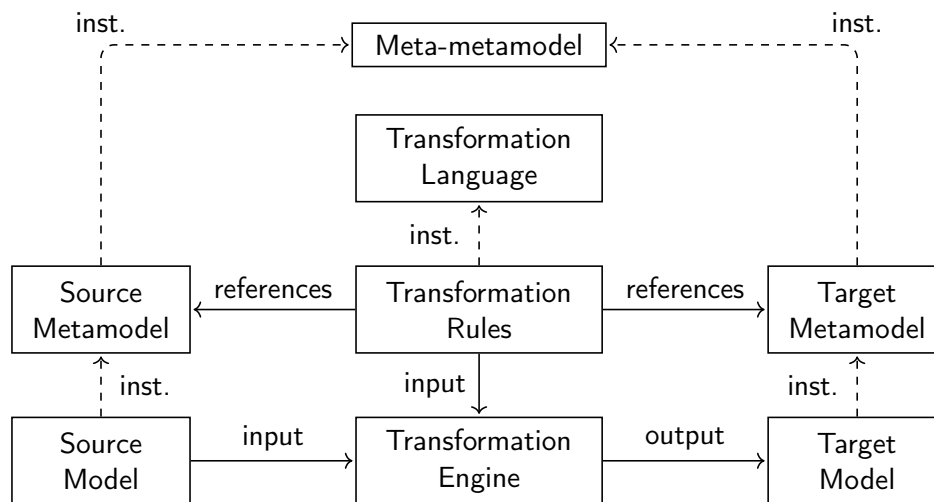
<sup>1</sup>Meta-Object Facility

<sup>2</sup>Essential MOF

<sup>3</sup>Complete MOF

<sup>4</sup>Extensible Markup Language

<sup>5</sup>XML Metadata Interchange



**Figure 2.3:** An overview on model transformation based on [16]

If they are the same it is called endogenous. Code generation is an example of an exogenous transformation while refactorings are endogenous.

- If the abstraction level changes from source to target model, the transformation is called vertical [17, p. 132]. Otherwise it is called a horizontal transformation. Refinements are a typical example of vertical transformations, while refactorings are usually horizontal transformations.
- Finally, transformations can be divided into syntactical and semantic ones [17, p. 133]. Syntactical transformations only use the source-model's syntax to produce the target-model while semantic transformations also consider the semantics of the model. Parsers which transform a concrete syntax into an abstract one classify as syntactical transformations.

There are many different languages that can be used to describe model transformations. These languages can be roughly divided into two groups: declarative and imperative languages.

Declarative languages describe *what* should be transformed by expressing relations between a source and a target model [17, p. 139]. Since the transformation is expressed in relations, it can often be used in both directions. Such a transformation is then called bidirectional. Common declarative languages are QVT<sup>1</sup>-Relational [18] or triple graph grammars.

If declarative languages are not sufficient to describe a transformation, imperative languages become necessary. With imperative languages the execution order of transformations can be explicitly defined, which is not possible with declarative languages. Therefore, they focus on *how* the transformation needs to be executed [17, p.139]. Any imperative programming language like Java or Xtend can be used to perform model transformations.

<sup>1</sup>Query, View, Transformation



There also exist hybrid languages like the Atlas Transformation Language [19] which combine both declarative and imperative features.

When transformations are used to keep two models consistent, it is often necessary that they are bidirectional so that changes in both models can be propagated to the other. Foster et al. [20] define the well-behavedness of bidirectional transformations which they call lenses. This lens concept was originally developed to handle view-update problems between a concrete model and a simplified abstract view. However, the concept can be applied to models at the same abstraction level accordingly. Each lens defines the transformation between two models and consists of two partial functions - one for each direction of the transformation - called GET and PUT. The GET function is used to extract an abstract view from a concrete model and the PUT function takes an updated view and applies the changes to the original concrete model. Getting a view from a concrete model and putting it back without modifying it should result in the same original model. This is called the GETPUT rule. Furthermore, getting an abstract view after putting a modified view into a concrete model should result in the same original view. This is called the PUTGET rule. Lenses are considered to be well-behaved if they obey both the GETPUT rule and the PUTGET rule. The lenses approach only allows modifications in the abstract view. However, the concepts above can be applied to scenarios in which both models can be modified. In such a scenario, every bidirectional transformation can be seen as two lenses such that each of the two models is the abstract view in one of them and the concrete model in the other lens. In chapter 4, we will refer to the PUTGET rule in terms of these adopted lenses.

### 2.1.5 Consistent and view-based modeling

When working with multiple heterogeneous models in a project, the represented information is fragmented and spread across these different models [3]. Relations between these models are usually not expressed explicitly. Moreover, some pieces of information can occur in more than one model. This redundancy can lead to inconsistencies which result in additional effort for the user if they try to keep the models consistent manually.

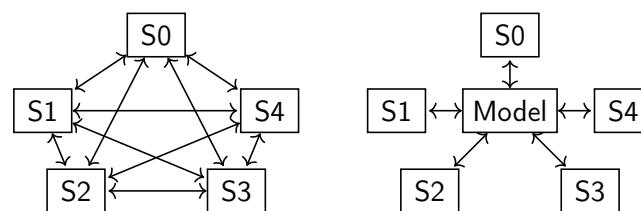
View-based development is an approach to cope with these challenges. The core idea is to describe originals using multiple models and granting the users access through different views [21]. These views are tailored to the specific requirements and rights of a certain user group. All interactions with the models - especially modifications - are not performed directly on the models but only on a user's view. After modifying a view, the changes are propagated to the underlying models to reach a consistent state. In this way, views hide redundancies and omit information that is irrelevant to the user.

To be able to talk about the consistency of models, a general concept of consistency is required. Since there is no uniform understanding of the term 'consistent', it is difficult to

define such a consistency concept formally. In this thesis, we will use an inductive definition of consistency. We will specify a set of consistency preservation routines which restore consistency after a model has been modified. These routines will result in consistent models iff the original model was already in a consistent state. In this way, redundancies that are not treated by consistency routines may lead to ‘implicit’ or ‘semantic’ inconsistencies. However, the models are still considered consistent in terms of the inductive definition.

In order to keep the underlying models consistent, incremental transformations can be used to restore consistency after a view was modified. Changes can be computed by comparing the state before and after a modification. However, this state-based approach suffers from information loss: e.g. moving a model element cannot be distinguished from deleting and recreating it. Alternatively, changes can be obtained by monitoring the modifications performed on a view which constitutes a delta-based approach [22]. In the latter approach, transformations are based on a sequence of changes, a so-called delta. If it is not possible to monitor a view, deltas can often be derived from a comparison of two model states. However, these deltas suffer from the mentioned information loss. Incremental change based consistency maintenance works in an inductive way: performing changes in a consistent state will result again in a consistent state, as explained above.

Approaches for creating views can be divided into two categories [23]: synthetic and projective approaches. In the synthetic approach, developers create views and specify relations between those views. In this way the model is defined by these different views (see the left graph in figure 2.4). The projective approach assumes an already existing model. All required views are derived from that model in some form of model transformation (see the right graph in figure 2.4).



**Figure 2.4:** Synthetic views (left) and projective views (right) based on [24, fig. 1]

The Orthographic Software Modeling approach by Atkinson, Stoll and Bostan introduces the idea of a Single Underlying Model [25]. Projective views can be derived from this SUM. In this way, each view has to be kept consistent only with this one SUM. Thus keeping the views consistent requires low effort, but the OSM<sup>1</sup> approach has its drawbacks when it comes to creating the SUM. Defining a single metamodel to capture all the information required for all views is far from easy when facing a complex domain. This problem was tackled in the VITRUVIUS approach.

<sup>1</sup>Orthographic Software Modeling

### 2.1.6 Vitruvius

The VITRUVIUS approach on view-based development by Kramer, Burger and Langhammer [3] carries on and evolves the ideas of the OSM approach [25]. The ideas of accessing models through views and central information maintenance are adopted. In this way, it preserves the advantages of projective views. However, a monolithic single underlying model is hardly viable (see section 2.1.5) so VITRUVIUS uses a VSUM<sup>1</sup>. To create a VSUM metamodel, multiple metamodels can be combined in a non-invasive way by specifying consistency preservation rules between them. Thus, a VSUM (an instance of a VSUM metamodel) is composed of multiple models which in combination describe the original. This constitutes a synthetic approach. Since VITRUVIUS combines projective and synthetic concepts, it is a hybrid approach.

A VSUM consists of multiple models containing redundant information. Therefore, keeping them consistent takes more effort than the monolithic SUM<sup>2</sup> in the OSM approach. Changes to the views are recorded and passed to the VSUM after a certain period of time, upon saving the view or when explicitly triggered by the user. Then, consistency preservation rules are executed to restore consistency between the VSUM's models.

VITRUVIUS provides two languages to define consistency conditions: the *Mappings* language and the *Reactions* language [26]. Mappings are used to define declarative semantic relations between elements (including metaclasses, attributes and references) of different metamodels in the VSUM metamodel. Some changes to the views result in inconsistencies that cannot be resolved by a mapping definition. To cope with these changes, reactions can be defined which trigger specific routines when observing specific changes. These routines are imperative descriptions of how to restore consistency after the modification. Every reaction reacts to a specific change. VITRUVIUS allows to define reactions for creating or deleting model elements of a certain type, replacing attribute values or references as well as inserting, removing or replacing elements in a list. VITRUVIUS selects the appropriate reaction depending on the current change.

---

```

reaction ReactToSomeChange {
    after element model_1::ElementB created and inserted in
        model_1::ElementA[listOfBs]
    call {
        someRoutine(affectedEObject, newValue)
    }
}

```

---

**Listing 2.1:** An exemplary VITRUVIUS reaction

---

<sup>1</sup>Virtual Single Underlying Model

<sup>2</sup>Single Underlying Model

Listing 2.1 shows an exemplary reaction which triggers the routine `someRoutine` whenever a new `ElementB` (`newValue`) is created and inserted into the list `listOfBs` of some `ElementA` (`affectedEObject`).

Routines consist of a `match`-block and an `action`-block. Within the `match`-block, model elements corresponding to the routines input parameters can be retrieved. Furthermore, preconditions can be specified which have to be fulfilled, otherwise the routine will not be executed. The `action`-block describes the actual process for restoring the consistency. It may include creating, updating or deleting model elements as well as adding or removing correspondences between them. Furthermore, code-block containing xtend-code can be used with the `action`-block. Listing 2.2 shows an exemplary routine which takes two inputs `a` and `b` which are both model elements of `model_1`. In the routine's `match`-block, the model element `c` corresponding to `a` is retrieved and the `check`-statement defines a precondition. The routine's `action`-block is executed unless the corresponding element `c` does not exist or `c.hasProperty` evaluates to *false*. In case those preconditions are met, a `model_2` element `d` is created and initialized. After that, the two elements `b` and `d` are declared correspondent and element `c` is updated. Finally the routine calls another routine before it terminates.

---

```
routine someRoutine(model_1::ElementA a, model_1::ElementB b) {  
  match {  
    val c = retrieve model_2::ElementC corresponding to a  
    check c.hasProperty  
  }  
  action {  
    val d = create model_2::ElementD and initialize {  
      d.attribute = b.attribute  
    }  
  
    add correspondence between b and d  
  
    update c {  
      c.listOfDs.^add(d)  
    }  
  
    call {  
      someOtherRoutine()  
    }  
  }  
}
```

---

**Listing 2.2:** An exemplary VITRUVIUS routine

VITRUVIUS uses a change-driven approach, which makes it difficult to integrate legacy models. To be able to ensure consistency by induction, a consistent initial state is required, which is usually not given with legacy models. In order to still be able to work with legacy models, two strategies have been developed [27, 28]: The construction of a given model can be simulated, which only allows one legacy model. Alternatively, an arbitrary number of legacy models can be joined. To do so, correspondences between those models have to be established, which takes more effort.

## 2.2 Transport studies

Transportation in general can be studied on multiple levels of abstraction each targeting different aspects of transport and traffic. Transport planning and traffic engineering mostly use models to predict the performance of transport facilities. Usually transport planning models describe transportation at a larger scope than traffic engineering models. To provide an overview of the usage of these models and their differences, we will give a short introduction to transport planning and traffic engineering in sections 2.2.1 and 2.2.2.

As mentioned in section 2.1.1, the model concept varies across different scientific fields. Even within the field of transport studies the term *model* is used with multiple meanings: for example dynamic driving behavior models or the static network models. In this thesis, we will use the term *model* to refer to the static representation of networks. Furthermore, we will work with *microscopic* traffic simulation programs which consider individual vehicles as opposed to *macroscopic* simulation programs which aggregate vehicles to characteristic values like traffic flow.

In this thesis, we will develop and compare the *metamodels* of the microscopic traffic simulation programs PTV VISSIM and SUMO before identifying relations between them. These relations will then be used to develop model transformations to keep PTV VISSIM and SUMO *models* consistent. Sections 2.2.4 and 2.2.5 give a short introduction into both tools.

### 2.2.1 Transport planning

Traffic arises due to the fact, that people want to pursue activities (like living, working, shopping, etc.) at a location where they are currently not located [29, p. 205]. This generates a transport demand which has to be met by some sort of transport supply. This supply may include infrastructure for private transportation like roads, intersections, foot and bicycle paths or parking spaces as well as public transportation infrastructure like rails, bus stops or bus lines and their schedules. If there exists a mismatch between the current transport demand and the required transport supply, transport planning models and methods can be used to develop and evaluate measures to overcome such a mismatch [29, p. 1].

Carrying out these measures is usually expensive and the involved infrastructure has a long lifespan [29, p. 1]. Thus, it is important to know the effects of a certain measure before it is implemented. Transport planning models usually have a wide scope (e.g. an entire city or wider) to predict the effects not only in the planning area where the measure is implemented, but also in the extended assessment area which is affected by the measure [29, p. 7].

### 2.2.2 Traffic engineering

In transport planning, the main goal is to understand how traffic comes to be and how it will react to changes in transport supply. Traffic engineering instead asks the question: Now that there is traffic, how can it be handled efficiently and safely? [30, p. 17f] In order to model the traffic flow realistically, the elements of the traffic network have to be modeled in greater detail. Due to this increase in detail compared to transport planning models, it becomes infeasible to model e.g. a whole city. Therefore, the traffic flow is usually modeled on a smaller scope containing only a few junctions, a train station or a motorway access. Also, traffic engineering measures usually have a smaller spatial and temporal scope than transport planning measures. So it is not only infeasible to model a whole city in such detail but also usually not necessary.

Traffic engineering models are used to model choices of drivers during their trips like the selection of their desired speed, the selection of a lane or the car following distance [31, p. 51f]. To model for example the lane selection of vehicles, those lanes have to be modeled explicitly in the network. Thus a high degree of detail is required in traffic engineering domain models.

These dynamic models can be applied in a simulation to simulate the behavior of vehicles. The resulting traffic can be used to estimate the quality of the existing or planned road network. A typical workflow of a traffic engineer starts with the construction of a model of the existing infrastructure and demand. This model needs to be calibrated until it matches the currently observed traffic [31, p. 1]. Finally, the model's transport supply can be modified to represent the planned state. Since the original model was calibrated, it can now be used to estimate the traffic in the planned state.

### 2.2.3 Coupling macroscopic and microscopic models

The idea of linking macroscopic and microscopic models has been around for several years. In their paper [32] from 2001 Fellendorf, Friedrich and Vortisch present a set of techniques for mapping a given macroscopic PTV VISUM model to a microscopic PTV VISSIM model. To estimate the quality of individual intersections or to plan the program of traffic lights it suffices to consider a subnetwork with only a few intersections. To perform microscopic simulation on that subnetwork, the selection in the macroscopic network has to be mapped to a microscopic network which especially models the network's geometry in more detail. If the macroscopic network was imported from a digital road network (e.g. Open Street Map [33]), its geometry is already a rough estimation of the actual road network.

Links (between intersections) can be mapped by using the imported geometry and automatically estimating the number of lanes based on the link's capacity, length and speed-limit.

The mapping of (at-grade) intersections is more complex. Although the macroscopic model contains a set of abstract turning movements (in the scope of links), the microscopic model requires the geometry of junctions on lane level. Therefore, the user has to provide additional topological information like the number of lanes at the crossing, their linkage, their width and the length of the waiting lane. With this additional information the junction's geometry can be generated automatically. When working with larger networks intersection-templates allow for a faster transformation. These intersection-templates can also be used for grade separated intersections although their geometry is usually already modeled by the imported road network.

After mapping an intersection's geometry, its traffic regulations needs to be mapped as well. There are three types to consider: priority to the right and priority road and signal control.

Aside from the road network which is required to simulate private transport, public transport infrastructure has to be mapped as well. This includes stops and public transport lines as well as a schedule. In the microscopic model, the actual stopping positions are required since most stops have at least two, one in each direction. These stopping positions, often given in geo-coordinates, have to be assigned to the corresponding link manually if the coordinates diverge from the modeled road.

These mapping techniques still require some manual effort and they do not allow transformations in both directions. However, they do point out which aspects of the domain are important. Hence, we will especially consider infrastructure and the geometry of links and lanes, traffic regulations and public transport when creating the metamodels and comparing them.

### 2.2.4 PTV Vissim

PTV VISSIM is a microscopic, behavior-based multi-purpose traffic simulation program [34, p. 63]. It is a commercial tool for traffic engineers developed at PTV<sup>1</sup> and can be used to simulate urban and motorway traffic, private and public transport as well as pedestrian movements.

To be able to simulate traffic flow, the transportation supply system has to be modeled using a graphical editor. In PTV VISSIM, this is divided into 4 building blocks [34, p. 67]. The first block contains elements, which model the static, physical infrastructure like roads and railways, sign and signal posts or parking lots and public transport stops. Secondly, the traffic (not the traffic flow) on that infrastructure has to be modeled. This includes vehicle specific features, routing decisions and public transport lines. Finally, the traffic control block contains rules to control the traffic at intersections like priority rules or signal programs. The fourth block does not contain elements to model the transportation supply system but rather

---

<sup>1</sup>PTV Planung Transport Verkehr AG



output data produced by the simulation like the animation of cars or statistical data collected during the simulation.

The first three blocks provide a good overview of the elements that should be considered when comparing the PTV VISSIM and the SUMO model.

After the transportation supply system is modeled, the actual traffic flow can be simulated. Therefore, PTV VISSIM includes several (dynamic) models to simulate the longitudinal and lateral movement of cars. The psycho-physical car-following models *Wiedemann 74* [35] and *Wiedemann 99* [36] describe longitudinal movements. To describe lateral movements, a lane selection model and a lane changing model is used. Since these two models do not allow the driver to plan ahead, PTV VISSIM extends these models by some tactical driving behavior models like anticipated driving at conflict areas and cooperative merging. To distribute the vehicles over different routes, PTV VISSIM offers three options. Fixed routing distributes vehicles at routing decision points to one possible route. This can be used for intersections where the distribution of turning movements is known. Dynamic routing allows route guidance systems to dynamically reroute vehicles based on detector values. Finally, PTV VISSIM includes dynamic assignment which is an iterative process in which vehicles are more likely to select a route with less estimated travel time (using a Kirchhoff distribution). This iterative process converges towards an equilibrium. Additionally to the models describing vehicle movements, PTV VISSIM uses the Social Force model by Helbing and Molnár [37] to simulate pedestrian movements.

### 2.2.5 SUMO - Simulation of Urban Mobility

SUMO is an open source road traffic simulation tool developed at the German Aerospace Center (DLR). The intention behind SUMO is to provide a free and extensible simulation program to unify the landscape of different small (incomplete) simulations developed in various theses [34, p. 269ff]. Those tools were often developed individually and lost support after the thesis was published. SUMO allows to compare different models and algorithms on the same basis.

SUMO is not only a simulation tool, but also provides assistance in setting up the traffic network. Networks can be imported from other simulation tools like VISUM and PTV VISSIM or from other formats like Open Street Map. SUMO then applies heuristics to compute missing values. Additionally, travel demand can be imported in form of OD-matrices which is then converted into the SUMO format: vehicle trips which consist of a departure time, a starting position and a destination. These trips do not include the routes on which the vehicles will travel. These routes are computed by some traffic assignment algorithm, e.g. a shortest paths algorithm or based on observed turning probabilities. Furthermore, a dynamic traffic assignment can be computed using the approach developed by Gawron [38]. It is based

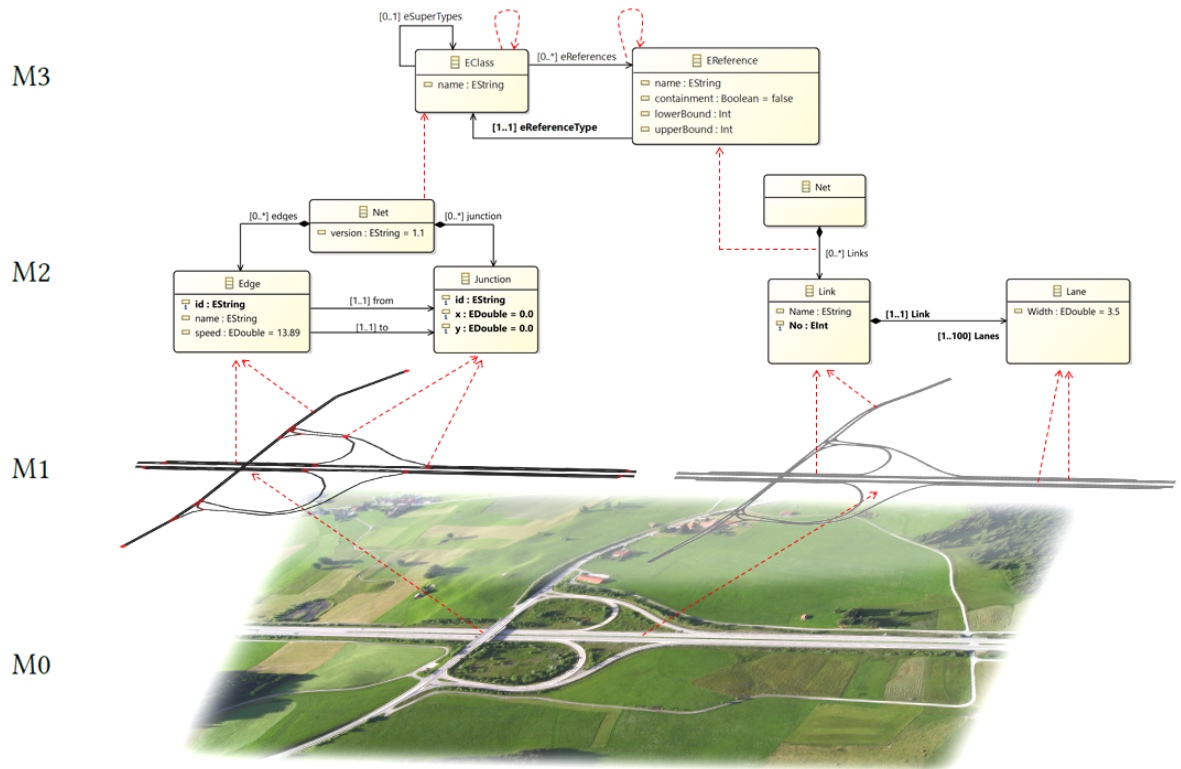
on the simulated travel times of the previous iteration and converges towards an equilibrium. Alternatively an iterative logit model can be used or the incremental oneShot assignment which calculates the fastest paths for each vehicle at its departure time.

To model longitudinal movements of vehicles, SUMO uses a model based on the car-following model developed by Krauß [39]. It was designed to allow fast computation of the vehicles movement and includes random driver imperfection to model spontaneous jams. The lane changing behavior is described by a model developed by Erdmann [40]. Additionally SUMO introduced a car-following and lane changing API to include other models in SUMO [41]. For example, SUMO now also provides the *Wiedemann99* model.

As mentioned above, SUMO does provide the possibility to convert a PTV VISSIM model into a SUMO model [42]. This transformation includes the road infrastructure, signal programs, routes and detectors. This covers the basic aspects required for a traffic simulation, however, this transformation is not suitable for keeping PTV VISSIM and SUMO models consistent. To do so, a bidirectional transformation is required that can handle incremental changes to either model.

### 3 Metamodels

In this chapter, we will develop two metamodels that represent the structure of PTV VISSIM models or SUMO models respectively. These metamodels serve as a basis upon which correspondences and consistency rules for concrete PTV VISSIM and SUMO models can be defined (see chapter 4).



**Figure 3.1:** The four modeling layers *M0-M3* applied to PTV VISSIM and SUMO models with exemplary instance-of relations (dashed arrows). *M0* is based on [43]

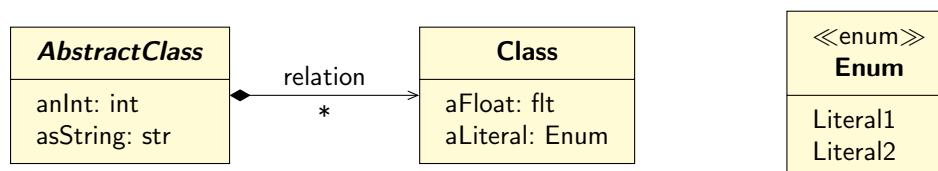
Figure 3.1 shows how PTV VISSIM's and SUMO's *models* and *metamodels* can be assigned to the four modeling layers described in section 2.1.2. Layer *M0* contains an exemplary motorway junction which is the original in this example. That original is represented by a SUMO model on the left and a PTV VISSIM model on the right at layer *M1*. These two models in turn are instances of the SUMO metamodel or the PTV VISSIM metamodel at layer *M2* respectively, which will be developed in this chapter. Layer *M3* contains the self-describing meta-metamodel *Ecore*.

The following sections are divided into four domains: infrastructure, traffic control, traffic demand and simulation output each of them containing multiple subdomains. For every subdomain, we will describe the relevant elements of both PTV VISSIM and SUMO and explain some of their dynamic semantics that are important to know when defining relations between them. In this thesis we will not consider aspects of PTV VISSIM and SUMO that are concerned with the visualization of the model elements in the simulation. Furthermore we will focus on models of vehicle traffic and therefore ignore PTV VISSIM's and SUMO's pedestrian simulation.

Since both PTV VISSIM and SUMO view vehicles and their drivers as one unit, we will use the term *vehicle* to refer to properties of both the driver and the vehicle.

PTV VISSIM and SUMO both persist their models in an XML format. They also both provide XSD<sup>1</sup> files which describe the structure of their XML files. In terms of metamodels, the XML schemas are a definition of a concrete syntax. The eclipse modeling framework (EMF) allows to generate *Ecore*-models from XSD files. However, the metamodels derived from PTV VISSIM's or SUMO's XSD files are closely coupled to the concrete XML syntax and contain various redundancies. This is why the metamodels used in this thesis were reverse engineered from the PTV VISSIM and SUMO documentation as well as their XML schema definitions. In the case of PTV VISSIM, the metamodel is based on its user manual [1] as well as the documentation of its COM interface [44]. The interface documentation is almost complete in terms of classes and their attributes, however, it sometimes lacks a description of their semantics. Since SUMO is an open source project which has been extended by various people, its documentation [2] does not explain the semantics of all attributes entirely. Furthermore, the documentation is inconsistent at some points. Some model elements like connections are described at multiple places with different sets of attributes which are inconsistent in terms of mandatory and optional attributes [45, 46].

In the following sections, the abstract syntax of PTV VISSIM's and SUMO's metamodels are described in the form of class diagrams. For both PTV VISSIM and SUMO, an XML-based concrete syntax is already given by the input format that both programs accept. The context constraints and the dynamic semantics of both metamodels will be given in natural language text. We will use a notation for the abstract syntax that is based on the notation of EMF's *Ecore* diagram editor.



**Figure 3.2:** The used metamodel notation based on the notation of the EMF-Ecore editor.

<sup>1</sup>XML Schema Definition

---

Figure 3.2 shows an example of the used notation. Classes have a name and a list of attributes. If a class is abstract, its name is given in italics. Every attribute has a name followed by one of the following types: string (str) for textual attributes, integer (int), boolean (bool) or float (flt) for a floating point number. Furthermore, enumerations (enums) can be the type of an attribute. In that case, the attribute holds one of the enumeration's values.

## 3.1 Modeling Transport Infrastructure

Transport infrastructure is required to support people and goods in traveling from one place to another. In the following sections, we will look at basic infrastructure elements like roads and intersections in section 3.1.1, some elements like traffic signs and signal heads that indicate traffic rules in sections 3.1.2 and 3.1.3 and finally in section 3.1.4 we discuss parking lots and public transport stops.

### 3.1.1 Roads and Intersections

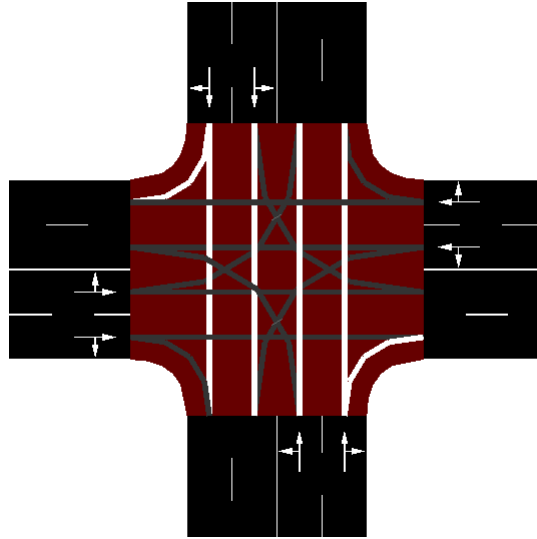
The basic infrastructure elements that enable transport are roads. They connect two points and allow people to travel from one point to the other. Where roads meet, they form intersections at which people usually can choose between multiple roads that lead away from the intersection in different directions. The representation of these elements forms the basic building blocks of PTV VISSIM and SUMO which most other model elements are built upon.

#### 3.1.1.1 Roads and Intersections in SUMO

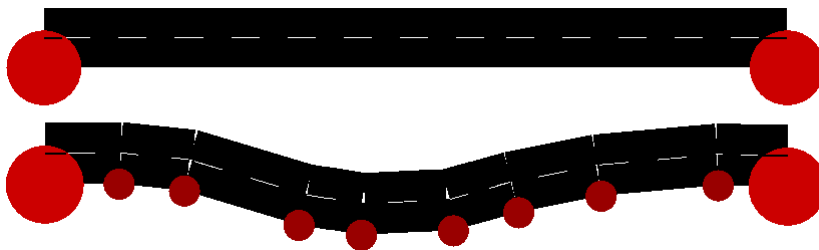
SUMO models the rough geometry of the road network using a directed graph consisting of edges and nodes. The nodes are called junctions and represent intersections where roads cross, merge or split while edges represent the road segments between them. Every edge consists of at least one lane which can allow or prefer certain vehicle classes to model multi-modal scenarios and distinguish cars, trams and bicycles.

At junctions, multiple edges meet each other, so the possible turning movements at crossings or the linkage of merging and splitting lanes need to be modeled. SUMO uses *Connection* objects for this purpose. Each connection links exactly two lanes. Figure 3.3 shows the connections at an exemplary intersection in SUMO where each incoming lane is connected with the one on the opposite side and one lane to its right or left.

Junctions, edges, lanes and connections each have a shape consisting of a set of points. Figure 3.4 shows an exemplary edge in SUMO once straight and once with a custom shape. In SUMO, the length of an edge is calculated based on its shape by default. However, in case an abstract geometry of the roads is used (e.g. only modeling the topology) but the actual distance between junctions is still required, the length of an edge can also be set to a custom value.



**Figure 3.3:** The connections at an intersection in SUMO each connecting two lanes. The connections on which vehicles get the right of way are displayed white. All minor connections are displayed in gray.



**Figure 3.4:** The graphical representation of edges in SUMO: one straight edge with two lanes connecting two junctions (above) and the same edge with a modified shape (below).

Figure 3.5 shows SUMO's metamodel elements that model the actual roads as mentioned above. In the following, we will list these elements with some of their properties:

- *Junctions* represent intersections of the road network and have an *x*, *y* and *z* coordinate. They have a unique *id* and can contain a set of *Positions* which models its *shape*. Each *Junction* also references all incoming *Lanes* (*incLanes*).
- *Edges* are defined by their *id*. Additionally, they can be assigned a *name*. Each *Edge* contains a set of at least one and at most 63 *Lanes*. Furthermore *Edges* have a *length* which is either a custom value or computed from its *shape* (a set of *Positions*).
- Each *Position* has an *x* and a *y* coordinate and can optionally have a *z* coordinate.
- *Lanes* have a unique *id*. They also have an *index* which indicates their position within their *Edge*. The right-most *Lane* of an *Edge* has the *index* 0. A *Lane*'s *id* is composed of its *Edge*'s *id* and that *index*. Each *Lane* has its own *width* and can reference *Vehicle Classes* that are allowed, disallowed or preferred to use the *Lane*. All other *Vehicle Classes* are not allowed to use it. The semantics of preferred *Vehicle Classes* is not given in the documentation of SUMO [2].
- A *Connection* ties together exactly two *Lanes* at a *Junction* and has a *shape* to model the geometry of their linkage. It references the *Lane* it originates from (*fromLane*) and the *Lane* it ends in (*toLane*) as well as the two *Edges* those *Lanes* are contained in. Furthermore, a *Connection* can allow different *Vehicle Classes* to use it than the *Lanes* it connects.

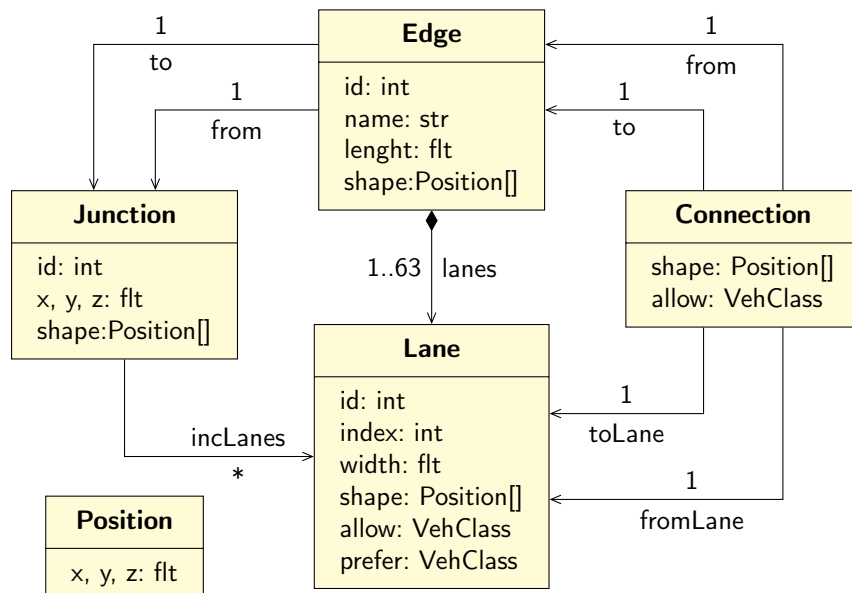
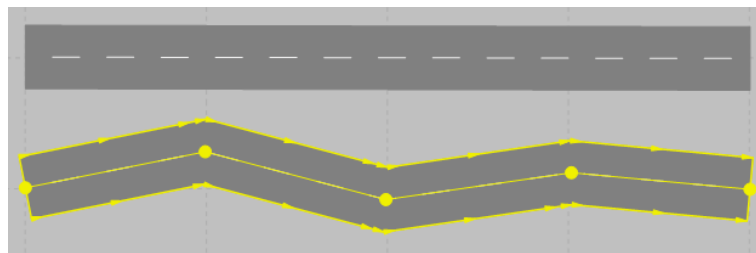


Figure 3.5: SUMO Metamodel: Roads and Intersections



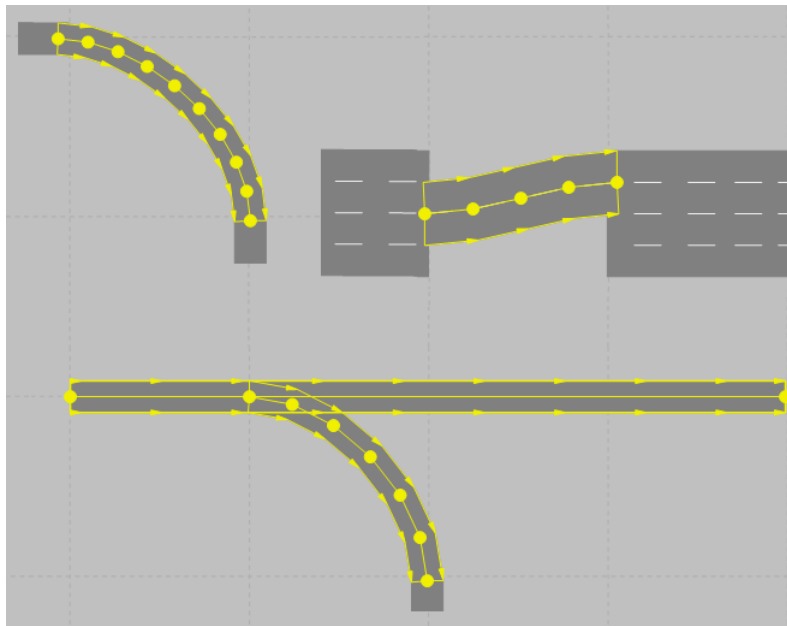
### 3.1.1.2 Roads and Intersections in PTV Vissim

Instead of modeling the road network using a directed graph that consists of nodes and edges, PTV VISSIM uses links as basic building blocks which do not model nodes explicitly to be more flexible [34, p. 68]. The geometry of a link is modeled as polygonal chain based on a set of 2D or 3D points. Figure 3.6 shows two links in PTV VISSIM both with two lanes. The second link has a custom shape and the points of the polygonal chain that define the shape are highlighted. PTV VISSIM allows to place links on different levels which assures that all links on one level have the same default z coordinate. This can be used to model multistory buildings. Since we will not consider the metamodel elements of pedestrian simulation in this thesis, the level of all links is assumed to be the base level. The geometry of links can still be three-dimensional if its points specify an offset from the link's level. Each road segment can be subdivided into different lanes. To model sub-networks reserved for certain modes of transportation like a railway network for trains or bicycle paths, each lane can block certain vehicle classes.



**Figure 3.6:** The graphical representation of links in PTV VISSIM: one straight link with two lanes (top) and the same link with a highlighted modified polygonal chain (bottom).

Since there are no nodes at which links can intersect, split or merge, PTV VISSIM uses connectors to model these traffic facilities. Connectors are special links that connect the lanes of two links. They can leave and join links at any position. Figure 3.7 shows three exemplary connectors. The two connectors on the top both start at the end of their links while the bottom one starts in the middle of its link. Like links, connectors are unidirectional so they can only connect lanes with appropriate directions. One connector can connect multiple pairs of lanes if they are adjacent to each other. Hence, a connector cannot connect the right and left most lane of a link, if there exist other lanes in between. For example, the top right connector in figure 3.7 connects the two middle lanes of the first link with the two leftmost lanes of the second link so it connects two adjacent lanes on both links. With these connectors, the possible turning movements at intersections and their geometry can be modeled in detail on lane level.



**Figure 3.7:** PTV VISSIM's connectors connecting two links. The connector on the right side connects two lanes of two four-lane links. The bottom connector starts in the middle of a link.

Figure 3.8 shows PTV VISSIM's metamodel elements that model the actual roads as mentioned above. In the following, we will list these elements with some of their properties:

- Each *Link* has a unique number (*no*) and can additionally be assigned a *name*. The length of a *Link* is calculated based on the referenced set of *Points* (*LinkPolyPts*) which model the geometry of the *Link*. Furthermore, *Links* have a *gradient* attribute to be able to model uphill or downhill slopes even if a planar road network is used.
- A *LinkPolyPoint* has an *x* and a *y* coordinate and can optionally specify a *z offset* along the *Z*-axis. This offset is measured relative to the *Link's Level*.
- *Lanes* can be identified by an *index* and the *Link* they are contained in. The right most *Lane* of a *Link* has the index 0. Each *Lane* has its own *width* and can reference multiple blocked *Vehicle Classes* that are not allowed to use it.
- Since *Links* are the basic building block in PTV VISSIM, a lot of other objects are placed onto them. More precisely, most objects are placed onto one of the *Lanes* contained by a *Link*. All those *Objects On Lanes* reference exactly one *Lane* they are placed on as well as a *position* on that *Lane*. That *position* is given as a distance relative to the *Lane's* beginning. Furthermore, every *Object On Lane* has a *name* and a number (*no*) that is unique in context of the concrete subclass.

- A *Connector* is a *Link*, that additionally references the *Lane* it originates from (*fromLane*) and the *Lane* it ends in (*toLane*). If a *Connector* connects multiple pairs of *Lanes*, these references refer to the right most *Lane* respectively. Since only adjacent pairs of *Lanes* can be connected by one *Connector*, the other connected *Lane*-pairs can be computed based on its number of *Lanes* and the *index* of the referenced *Lanes* in the origin/destination *Link*. The position along a *Link*, where a *Connector* leaves or enters a *Lane* is given as absolute distance from the *Link*'s beginning (*fromPos*, *toPos*). A *Connector* cannot connect *Lanes* that are contained in another *Connector*. In PTV VISSIM, *Links* and *Connectors* are implemented in one class and a Boolean attribute '*isConn*' indicates whether an object is a *Connector* or not. To separate these two concepts and to separate *Connector*-specific properties from the *Link* class we will model *Connectors* as a subtype of *Links*.

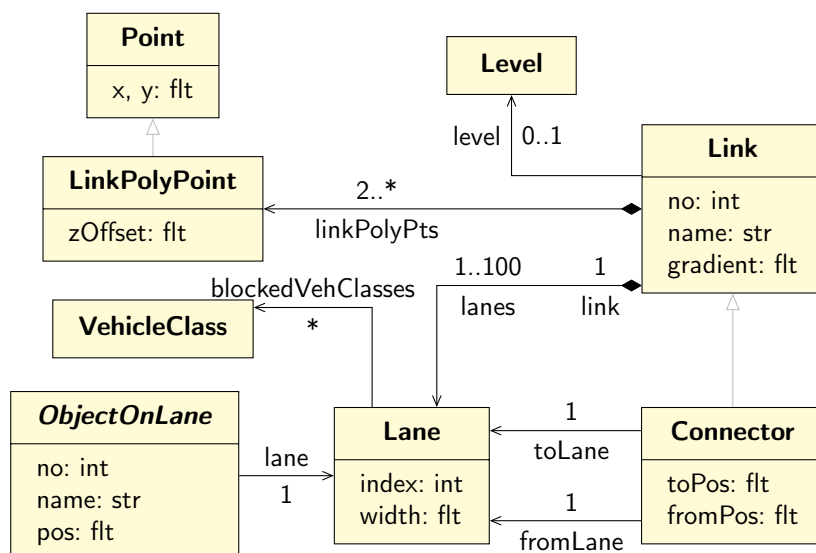


Figure 3.8: PTV VISSIM Metamodel: Roads and Intersections

### 3.1.2 Speed limits

In section 3.1.1 we introduced basic infrastructure elements like roads and intersections. On many roads, there exist certain rules to control the traffic such as speed-limits. These speed-limits define the maximum allowed speed on a road and therefore restrict people from drive with their desired speed. In urban areas, on roads that are under construction or on narrow roads the allowed speed can be restricted to ensure the safety of all drivers and pedestrians. Speed signs at the road side indicate the maximum allowed speed. Those signs are part of the infrastructure and in the following sections, we will look at their representation in SUMO and PTV VISSIM.

### 3.1.2.1 Speed limits in SUMO

In SUMO, every edge, lane and connection has a speed attribute, which sets the speed limit for the respective road segment. However, edges and lanes both having speed attributes would result in redundant information which is why only lanes receive a speed attribute in the metamodel. How vehicles react to the given speed limit - whether they respect it or decide to drive faster - is part of a vehicle's driving behavior (see section 3.3.2.1). Additionally to those static speed limits, SUMO provides *Variable Speed Signs* to model speed signs (e.g. on motorways) that only restrict the allowed speed at a certain time of day.

Figure 3.9 shows SUMO's metamodel elements that model the speed limits and variable speed signs. In the following, we will list these elements with some of their properties:

- Every *Lane* must define a speed limit given in its attribute *speed*.
- *Connections* can define a maximum *speed* for moving across the intersection.
- A *Variable Speed Sign* controls one or more *Lanes* and has a unique *id*. Furthermore, it contains a set of *Vss Steps* which define when to change the *speed* of the controlled *Lanes*. If a *Connection* does not specify a speed limit, the average speed limit of its origin and destination *Lane* is used.
- *Vss Steps* (Variable Speed Sign Steps) are pairs of *time* and *speed* values. The *time* value denotes the simulation time at which the speed limit will be changed to the given *speed* value. If a negative *speed* value is given, the speed limit is set to the original *speed* values of the controlled *Lanes*.

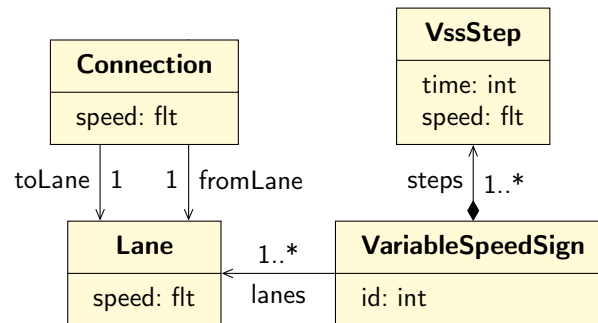
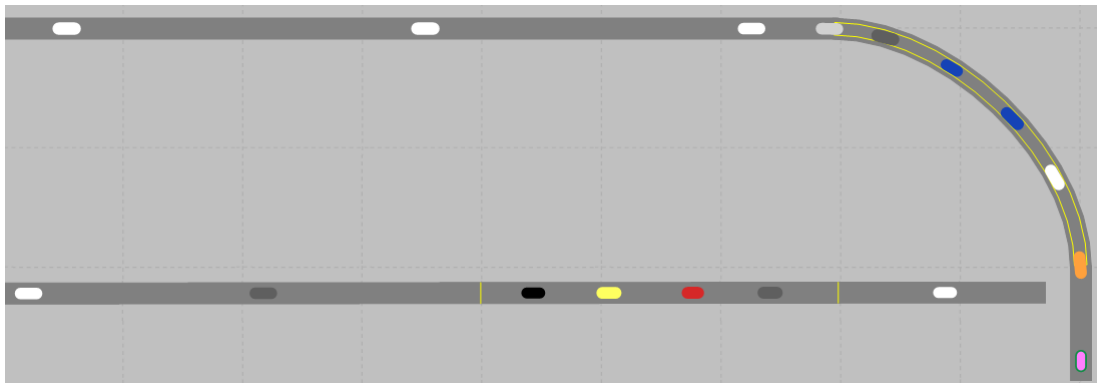


Figure 3.9: SUMO Metamodel: Speed Limits

### 3.1.2.2 Speed limits in PTV Vissim

PTV VISSIM does not model speed limits explicitly. Instead, the reactions of vehicles to a speed limit are modeled in the form of a distribution of their desired speeds. For example, motorways with a speed limit of  $120 \frac{\text{km}}{\text{h}}$  can be modeled by using a distribution with a mean of  $120 \frac{\text{km}}{\text{h}}$ . When vehicles are created during the simulation they are assigned a desired speed drawn from such a distribution. Since the speed limit can change along a vehicles

path through the network, PTV VISSIM allows vehicles to change their desired speed at certain points called desired speed decision. These desired speed decisions can be thought of as speed signs. As soon as vehicles pass these points they are assigned a new desired speed which is again drawn from a distribution. Alternatively, the desired speed can be reduced only for a certain area. This can be used to model the reduced speed in curves, construction sites on the road or traffic calmed areas. Figure 3.10 shows such a reduced speed area in the top right corner (yellow box) and two desired speed decisions (yellow lines) on the bottom link. Inside the reduced speed area and between the desired speed decisions the vehicles have a lower desired speed than outside.



**Figure 3.10:** A reduced speed area in a curve (top right) and two desired speed decisions (on the bottom link) where vehicles change their speed.

Figure 3.11 shows PTV VISSIM's metamodel elements that model speed limits. In the following, we will list these elements with some of their properties:

- A *Desired Speed Decision* is an *Object On Lane* that assigns new desired speeds to passing vehicles. Those desired speeds value are drawn from a *Desired Speed Distribution*. Since different kinds of vehicles can have a different distribution of desired speeds, *Desired Speed Decision* can reference multiple *Vehicle Class Desired Speed Distributions* to assign each *Vehicle Class* an individual distribution. Furthermore, a time period can be specified during which the *Desired Speed Decision* is active. This can be used to model variable speed signs that only restrict the speed limit at a certain time of day.
- *VehClassDesSpeedDistribution* objects model the ternary relationship between *Desired Speed Decisions*, *Vehicle Classes* and *Desired Speed Distributions*. They link exactly one *Vehicle Class* with exactly one *Desired Speed Distribution* and belong to exactly one *Desired Speed Decision*. In this way, if a vehicle of a certain *Vehicle Class* reaches a *Desired Speed Decision*, the correct *Desired Speed Distribution* can be determined. The vehicle's new desired speed will be drawn from that distribution.

- *Reduced Speed Areas* are placed on a single *Lane*, hence they are *Objects On Lanes*. They define a starting position on that *Lane* (see section 3.1.1.2) as well as the area's *Length*. Inside the *Reduced Speed Area*, vehicles will use a reduced desired speed drawn from a *Desired Speed Distribution*. Once the vehicles leave that area, they will continue to use their original desired speed. With this, short areas of reduced speed like construction sites on roads or tight curves can be modeled. Similar to *Desired Speed Decisions*, every *Vehicle Class* can be assigned an individual *Desired Speed Distribution*. This assignment of distributions to *Vehicle Classes* is modeled as *VehClassSpeedReduction* objects. *Reduced Speed Areas* can also specify a time period during which they are active.
- Similar to *VehClassDesSpeedDistribution*, *VehClassDesSpeedReductions* model the ternary relationship between *Reduced Speed Areas*, *Vehicle Classes* and *Desired Speed Distributions*. One *Vehicle Class* is assigned exactly one *Desired Speed Distribution*. With this, the correct distribution can be determined for all vehicles entering the *Reduced Speed Area*.

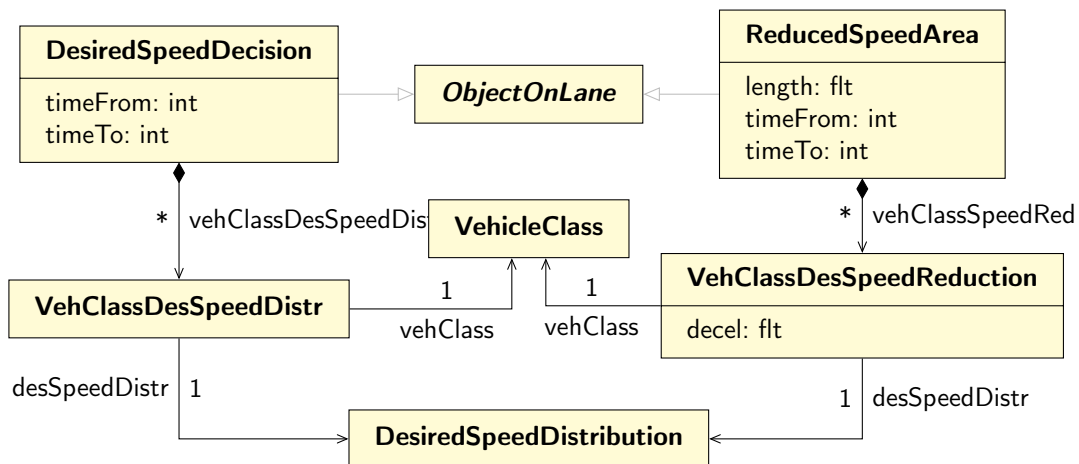


Figure 3.11: PTV VISSIM Metamodel: Speed Limits

### 3.1.3 Stop Lines at Intersections

Similar to the speed signs discussed in section 3.1.2 which control the traffic on roads, there are important rules that control the traffic at intersections. What those rules are can change from intersection to intersection and is therefore indicated by traffic signs. Those traffic signs or signal heads are part of the infrastructure. They also include stop lines, that mark positions on roads where waiting vehicles are supposed to stop.

### 3.1.3.1 Stop Lines at Intersections in SUMO

SUMO implicitly places signal heads and traffic signs at the entrance to junctions depending on how they are controlled. So the signal heads and stop signs themselves cannot be modeled explicitly. However, the stop line at which vehicles wait before crossing an intersection can be modeled explicitly. Every lane entering a junction can specify the position of such a stop line. Furthermore, multiple stop lines for different types of vehicles (e.g. to separate bikes and cars) can be defined.

Figure 3.12 shows SUMO's metamodel elements that model stop lines on Lanes and Edges. In the following, we will list these elements with some of their properties:

- *Lanes* can specify a set of *Stop Offsets* to model the stop lines for different *Vehicle Classes*. If a *Lane* does not specify a *Stop Offsets*, the stop line is placed at the beginning of the *Junction*.
- Every *Stop Offset* has an *offset value* which specifies the distance in meters between the stop line and the point where the *Lane* enters a *Junction*. To model different stop lines for different types of vehicles, a *Stop Offset* specifies which *Vehicle Classes* are affected by the stop line.

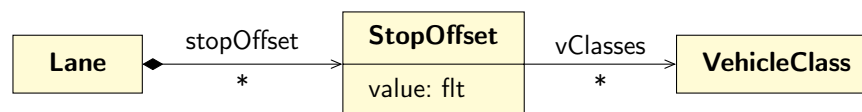


Figure 3.12: SUMO Metamodel: Stop Lines

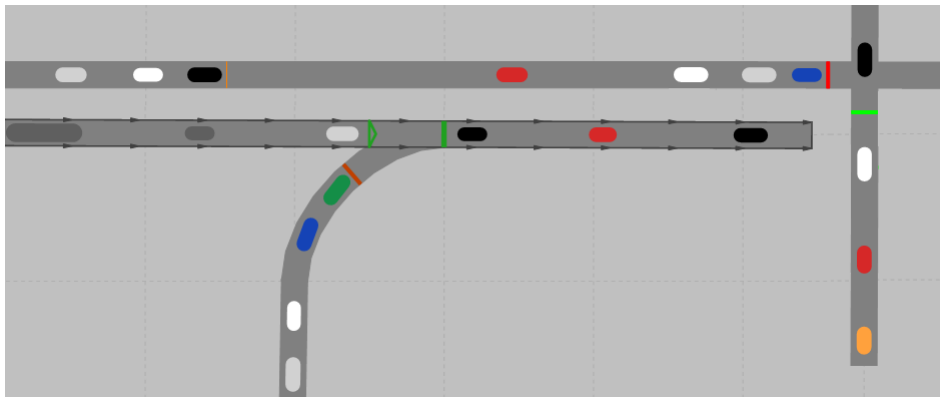
### 3.1.3.2 Stop Lines at Intersections in PTV Vissim

In PTV VISSIM stop signs, priority signs and signal heads are assigned to an individual stop line. These stop lines can be placed freely on any lane in the network. The stop lines of priority signs and signal heads can be defined for specific kinds of vehicles. This can be used to model multiple stop lines at a traffic light controlled intersection, so bicycles can wait closer to the intersection in front of the cars. Figure 3.13 shows these three kinds of stop lines.

In this section we will only look on the infrastructural aspects of these objects. How they are used to control the right of way will be discussed in section 3.2.1.2.

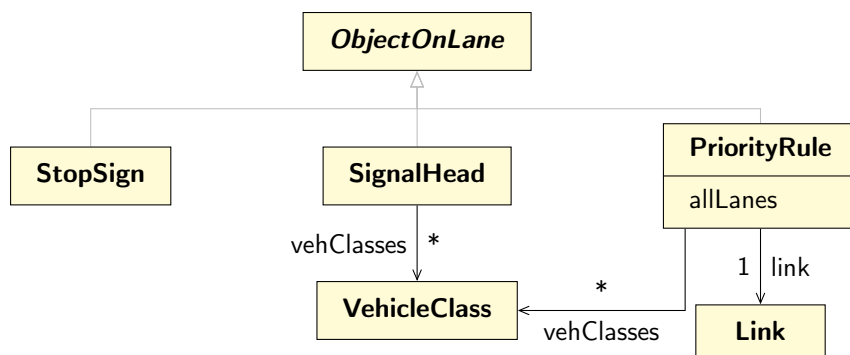
Figure 3.14 shows PTV VISSIM's metamodel elements that model stop lines. In the following, we will list these elements with some of their properties:

- *Stop Signs* are *Objects On Lanes* with a specific position on one *Lane* (see section 3.1.1.2). Vehicles must halt at the stop line for at least one simulation step.



**Figure 3.13:** Three different types of stop lines in PTV VISSIM: a stop sign (top left) in orange, a priority sign on the merging link in red and two signal heads at the crossing (top right) both displayed in the color of their current signal.

- *Signal Heads* display signals according to the program of a *Signal Controller* (see section 3.2.2.2). Like *Stop Signs*, each *Signal Head* is placed on one *Lane* at a position where the stop line is located. Vehicles will wait at the stop line until the *Signal Head* displays the green signal. *Signal Heads* can reference a set of *Vehicle Classes* to specify which kind of vehicles have to abide to the signal. With this, different *Signal Heads* and stop lines can be used for bicycles and cars.
- *Priority Rules* represent priority signs that are placed on *Lanes*. Vehicles have to wait at the stop line when there are passing vehicles on the major road. Similar to *Signal Heads*, *Priority Rules* can specify which kinds of vehicles (*Vehicle Class*) have to respect the priority sign. Additionally to the *Lane* a *Priority Rule* is placed on, it references the *Link* which contains that *Lane*. Instead of only covering a single *Lane*, the stop line can be extended across all *Lanes* of the referenced *Link* as indicated by the Boolean attribute '*allLanes*'.



**Figure 3.14:** PTV VISSIM Metamodel: Stop Lines



### 3.1.4 Public Transport Stops and Parking Lots

When vehicles drive through the road network, they usually do so because the drivers intend to pursue activities at some place they are currently not at. Once they have reached their destination, they interrupt their trips for a certain amount of time until they continue to drive to their next destination. Therefore, some infrastructure is required where drivers can park their vehicles. Similarly, infrastructure is required at which public transport vehicles can halt to pick up and drop off passengers. In this thesis we will look at parking lots for private transport vehicles and public transport stops for public transport vehicles.

#### 3.1.4.1 Public Transport Stops and Parking Lots in SUMO

SUMO supports four different types of stop infrastructure: parking areas, bus stops, container stops and charging stations. In this thesis we will exemplarily consider parking lots and bus stops since they both have a direct correspondence in PTV VISSIM. Both of them are located on a single lane and have a certain length along that lane. Parking areas can be located on the road, which leads to parking vehicles blocking the lane. Alternatively, parking areas can be located next to the road, which does not block the corresponding lane. Bus stops are located at the side of their lane and allow different public transport lines to pick up and drop off passengers.

Figure 3.15 shows SUMO's metamodel elements that model stop infrastructure. In the following, we will list these elements with some of their properties:

- *Stop Infrastructure* is the abstract supertype of all concrete stopping places and represents an arbitrary area where vehicles can interrupt their trips. Each concrete area is located on one *Lane*. On that *Lane* the area has a start and end-point (*startPos*, *endPos*), both relative distances to the *Lane*'s beginning. Furthermore, every *Stop Infrastructure* has a unique *id* as well as an optional *name*.
- A *Bus Stop* is a *Stop Infrastructure* where public transport vehicles can halt to pick up and drop off passengers. *Bus Stops* reference all *Lines* they are served by. Currently, these referenced *Lines* are used for visualization purposes in SUMO. If a *Bus Stop* is accessible from another *Lane* except the one it is located on, it contains a set of additional *Accesses*. Figure 3.17 shows an exemplary *Bus Stop* with such an additional *Access* on the next *Lane*.
- An *Access* is an additional possibility for public transport vehicles to enter a *Bus Stop* from another lane. It references a *Lane* and a position from where the *Bus Stop* can be accessed.
- A *Line* is a public transport *Line* that serves *Bus Stops*. Every *Line* has a *name* which is currently only used for visualization purposes in SUMO.

- A *Parking Area* is an abstract *Stop Infrastructure* that is either located on the road or next to it. These two concrete alternatives are realized by the subtypes *Road Parking Lot* and *Parking Space Area*. Figure 3.16 shows an example of both kinds.
- A *Road Parking Lot* is a *Parking Area* where the parking spaces are located on the road. The *width* and *length* of these parking spaces can be specified as well as their *angle* relative clockwise to the *Lane*. Additionally, the amount of parking spaces in that area can be specified (*roadSideCapacity*). The parking lot on the top *Lane* in figure 3.16 is an example of such a *Road Parking Lot*.
- A *Parking Space Area* is a *Parking Area* that is located next to the road. The area has a rectangular shape with a certain *width* and a length given by *startPos* and *endPos*. *Parking Space Areas* contain a set of *Parking Spaces* which model the individual parking positions explicitly. The bottom *Parking Lot* in figure 3.16 is an exemplary *Parking Area* containing multiple *Parking Spaces*.
- *Parking Spaces* belong to exactly one *Parking Space Area*. The position of a *Parking Space* is given by an *x* and *y* (and optional *z*) coordinate. Its size is determined by its *width* and its *length*. Additionally, *Parking Spaces* can be rotated by an *angle*. With this, the exact layout of a *Parking Space Area* can be modeled explicitly, while the parking spaces of *Road Parking Lots* are implicitly generated.

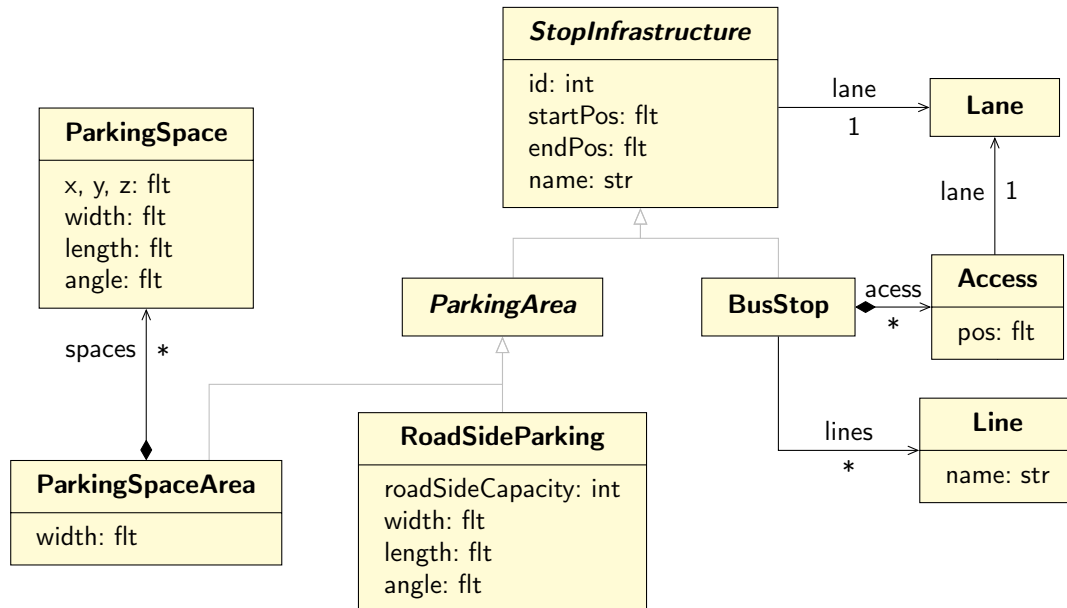
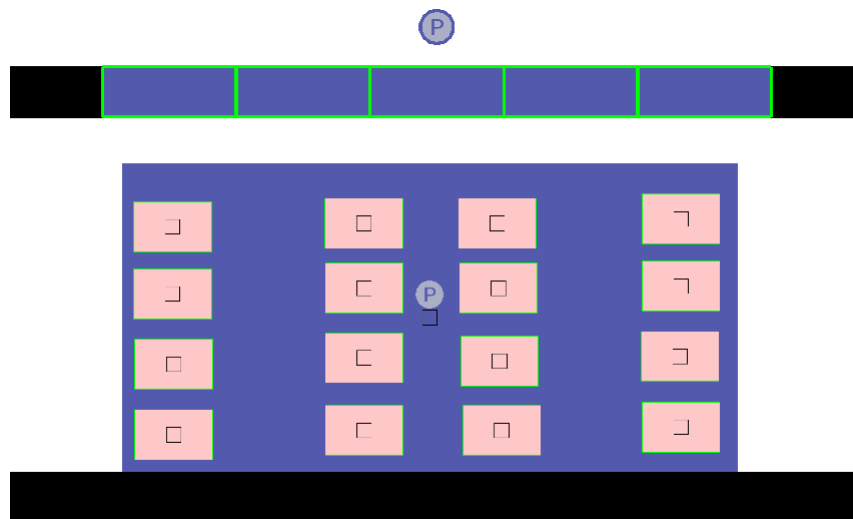


Figure 3.15: SUMO Metamodel: Public Transport Stops and Parking Lots



**Figure 3.16:** Two parking lots: one with parking spaces on the road (top) and one parking area next to the road.



**Figure 3.17:** A bus stop on the top lane and an additional access (green dot) to that stop on the bottom lane.

#### 3.1.4.2 Public Transport Stops and Parking Lots in PTV Vissim

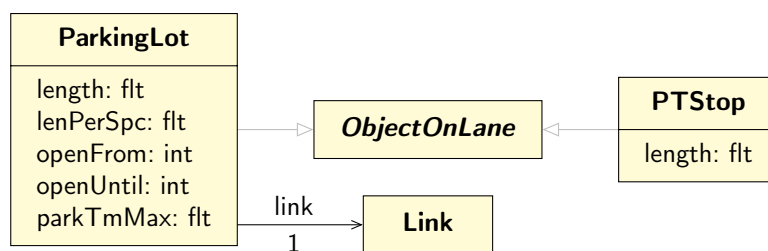
PTV VISSIM provides two infrastructure objects that allow vehicles to interrupt their trip through the network. Firstly, public transport stops can be used to model train or bus stops where public transport vehicles can halt to pick up and drop off passengers. Private vehicle can interrupt their trips by parking at a parking lot.

PTV VISSIM has a very detailed parking lot metamodel. For example the attraction of the individual parking spaces and parking costs can be modeled. Furthermore parking spaces can either be used as real parking lot with multiple parking spaces on which vehicles can park or as abstract parking lots that serve as start and end points for dynamically assigned routes. However, SUMO's metamodel of parking lots is much more simple which is why we will only look at some basic properties of PTV VISSIM's parking lots.

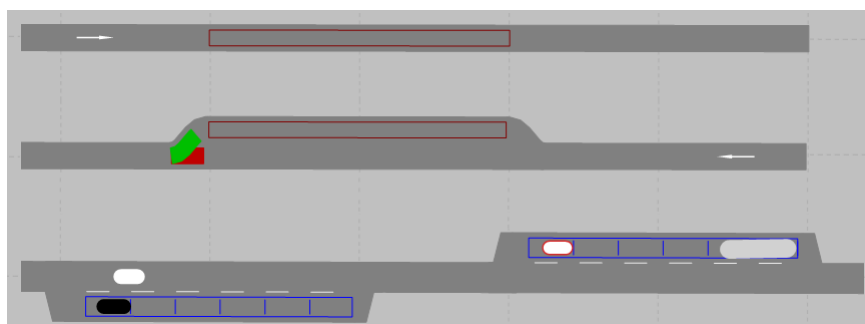
In this section we will only look on the infrastructural aspects of these facilities. The way they are used will be discussed in sections 3.3.2.2 and 3.3.3.2.

Figure 3.18 shows PTV VISSIM's metamodel elements that model parking lots and public transport stops. In the following, we will list these elements with some of their properties:

- *Parking Lots* start at a certain position on a *Lane* (see section 3.1.1.2) and have an individual *length*. Together, a *Parking Lot's* *length* and the length of each parking space (*lenPerSpc*) define the total number of parking spaces in the *Parking Lot*. Additionally to the *Lane* the *Parking Lot* is placed on, it references the *Link* which contains that *Lane*. Each *Parking Lot* can specify its opening hours *openFrom* and *openUntil*. Also, a maximum parking duration (*parkTmMax*) can be defined. The bottom *Link* in figure 3.19 contains two *Parking Lots* each consisting of six parking spaces.
- Like *Parking Lots*, *PTStops* are *Objects On Lanes*, which means they have a specific position on one *Lane*. They also have a *length* which determines the area in which public transport vehicles can halt. Figure 3.19 shows two exemplary *PT Stops*: one that is placed on the road (top *Link*) and one that is situated next to the road (middle *Link*) a so-called lay-by stop.



**Figure 3.18:** PTV VISSIM Metamodel: Public Transport Stops and Parking Lots



**Figure 3.19:** A public transport stop on the road (top link), a lay-by stop on the middle link and two parking lots on the bottom link.

---

## 3.2 Modeling Traffic Control

Intersections are critical elements in a road network [30, p. 141]. They are locations of increased traffic volumes and therefore influence the quality of transport on nearby roads. On the one hand, intersections have to treat the oncoming traffic efficiently so they can quickly continue their trips into the desired direction. On the other hand, vehicles should be able to cross an intersection safely. The number of vehicles that can cross an intersection in one hour is called the intersection's capacity.

Intersections can be divided into at-grade and grade separated intersections. On at-grade intersections, all vehicle streams move at the same level and can therefore cause conflicts. Grade separated intersections use multiple levels of height to avoid conflicting streams. In case of an at-grade intersection, the right of way has to be specified in order to assure a safe and efficient treatment of the oncoming traffic. In the following, we will look at different ways to control an intersection in section 3.2.1 and especially at signal-controlled intersections in section 3.2.2.

### 3.2.1 Right of Way at Intersections

In this section we will look at intersections that are not controlled by traffic lights. Depending on the traffic volumes towards the intersection it can be equipped with traffic signs indicating a certain right of way rule. If an intersection is not equipped with traffic signs, it is a common rule (in right-hand traffic) to yield the right of way to vehicles coming from the right side called right-before-left. If the capacity of an intersection with the right-before-left rule is not sufficient, other right of way rules can be used to increase its capacity.

For example, if the stream in one direction has a noticeably larger traffic volume than streams coming from the right and left, it can be prioritized so that the left and right stream yield the right of way and wait. The prioritized streams are also called main or major stream while the yielding streams are called waiting or minor streams. Vehicles in the minor stream wait until there is a sufficient gap between two vehicles in the major stream.

#### 3.2.1.1 Right of Way at Intersections in SUMO

In SUMO the right of way at a junction depends on its type. It can be based on the priority of incoming edges, controlled by a traffic light logic or follow the right-before-left rule. For each connection at a junction, the conflicting connections are computed to determine which vehicles have to yield their right of way. Figure 3.3 shows an exemplary priority crossing. The streams on the white connections get the right of way while vehicles on gray connections have to wait.

Figure 3.20 shows SUMO's metamodel elements that model the right of way at intersections. In the following, we will list these elements with some of their properties:

- The right of way at a *Junction* is determined by its *type*:
  - *unregulated*: All vehicles have the right of way. No collisions are detected inside the *Junction*.
  - *priority*: If a *Junction*'s *type* is '*priority*', vehicles may pass the intersection in decreasing order of their *Edge*'s *priority*.
  - *priority stop*: The right of way at the *Junction* is computed like a *priority Junction* with the additional rule, that vehicles on a minor connection always have to stop. This type can be used to model intersections where the minor roads have to obey a stop sign.
  - *allway stop*: All approaching vehicles have to stop before crossing the *Junction*. This type can be used for intersection with particularly poor visibility.
  - *right before left*: Vehicles give the right of way to vehicles coming from the right.
  - *traffic light*: The *Junction* is controlled by a *Traffic Light Logic* (see section 3.2.2.1). In case two conflicting streams get a green signal at the same time, the priority rules are used to determine the right of way. If the attribute '*uncontrolled*' of a connection is *true*, the *Junction*'s *Traffic Light Logic* will be ignored on that connection.
  - *zipper*: If a *Junction* is used to reduce the number of *Lanes* on a road segment (late merge) the type *zipper* can be used to swap the right of way at the merging point between the merging and the merged *Lane*.

If the *Junction*'s *type* is '*priority*' or '*priorityStop*' the right of way depends on the *priority* values of all incoming *Edges*. A *Junction*'s *Right Of Way Type* determines how the right of way is derived from these priority values: When using the '*default*' *Right of Way Type*, all incoming *Edges* are sorted by *priority*, *speed* and *number of lanes*. The first two *Edges* in that sorted list receive the right of way while all others become minor *Edges* and have to give way. When using the *Right Of Way Type* '*edgePriority*', the right of way is purely based on the *priority* values of the incoming *Edges*. If there are two conflicting streams with equal priority, the turning directions are considered. The Boolean attribute '*keepClear*' (of *Junction* and *Connection*) indicates, whether vehicles will try to avoid blocking the *Junction*. The result of the right of way computation is given as a set of *Requests* - one for each *Connection*. Furthermore, a *Junction*'s *visibility* defines the distance below which vehicles approaching the *Junction* are able to see the entire *Junction* as well as other approaching vehicles.

- *Roundabouts* require special treatment to determine the right of way. Therefore, a *Roundabout* references all *Edges* on the circle and the intermediate *Junctions*. Vehicles on these *Edges* will always get the right of way over vehicles entering the *Roundabout* at one of the *Junctions*.

- A *Request* describes how vehicles on one specific *Connection* will interact with streams on other *Connections* of that *Junction*. It references *Connections* (*response*) with a higher priority than the referenced *Connection* (*connection*). Vehicles on that *Connection* have to give way to those *response* *Connections*. Additionally, *Requests* reference all conflicting *Connections* (*foes*) which includes those with both higher and lower priority than the referenced *Connection*. Thus, the *response*-set is a subset of the *foes*-set. The attribute '*cont*' states, whether vehicles are allowed to drive past the stop line onto the *Junction* to wait there. This is typically used for vehicles turning left on the major road.
- A *Connection*'s *state* defines its individual traffic sign or rule:
  - On '*priority*' or '*priority stop*' *Junctions*, major *Connections* receive the state '*major*'. Minor *Connections* receive the state '*minor*' on '*priority*' *Junctions* and '*stop*' on '*priority stop*' *Junctions*.
  - On '*allway stop*' *Junctions*, all *Connections* get the state '*wait*'.
  - On '*right before left*' *Junctions*, all *Connections* get the state '*equal*'.
  - On '*zipper*' *Junctions*, the merging *Connections* get the state '*zipper*'. All other *Connections* get the state '*major*'.
  - On '*traffic light*' *Junctions*, all *Connections* get the initial state '*off*'. During the simulation, these *states* are set to the signals of the signal program's current *Phase*.

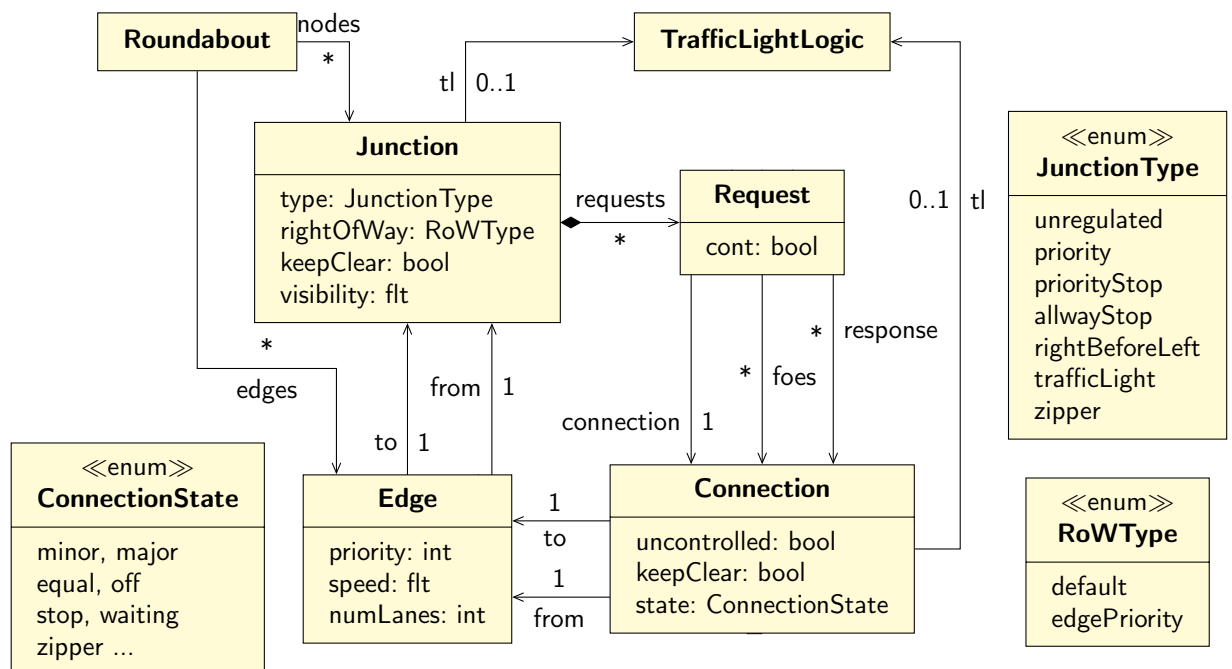


Figure 3.20: SUMO Metamodel: Right of Way at Intersections

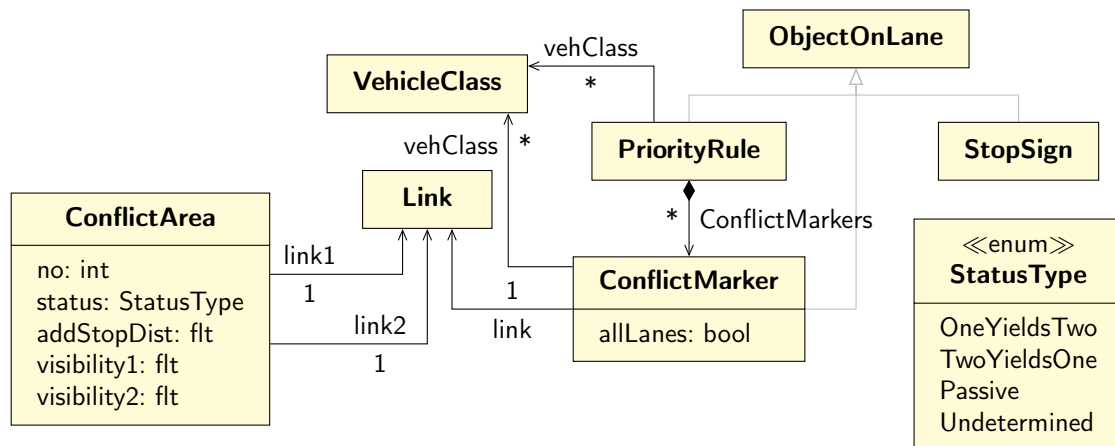
### 3.2.1.2 Right of Way in PTV Vissim

PTV VISSIM supports different ways for controlling the right of way at intersections. Conflict areas are the most basic specification of major and minor streams. Wherever two links intersect, a conflict area is created. At such a conflict area the right of way is always determined just for the two crossing links. To control the right of way across multiple conflict areas, priority rules can be used, which define a stop line where the minor stream must halt (see section 3.1.3.2) as well as a set of conflicting major streams. Vehicles on the minor stream have to wait at the stop line until there is a sufficiently large gap in those conflicting major streams. Additionally to the priority rule, a stop sign can be placed at the stop line, which forces vehicles to halt for at least one simulation step before looking for a gap. Another way for controlling the right of way across multiple conflict areas is using signal programs which is discussed in section 3.2.2.2.

Figure 3.21 shows PTV VISSIM's metamodel elements that model the right of way at intersections. In the following, we will list these elements with some of their properties:

- *Conflict Areas* are areas where exactly two *Links* intersect. Each of them can define a *visibility* which is the distance from the *Conflict Area* at which an approaching vehicle can see vehicles on the other *Link*. A *Conflict Area's status* describes which vehicles yield the right of way and wait. Figure 3.22 shows the four possible statuses. A *Conflict Area* can be *Passive* (both directions yellow). In this case, the streams ignore each other. Alternatively, one of the streams yields the right of way (the yielding direction is red the other is green). The fourth *Status Type* is *Undetermined* can be used for branching *Links*. Even though the streams are not conflicting, the vehicles should not ignore each other when driving in their desired direction but rather consider vehicles on both branching *Links*. Figure 3.23 shows such a situation where the gray vehicle on the *Conflict Area* recognizes the green one turning right and waits for it to leave the conflict area before turning left.
- *Priority Rules* define a stop line at which the yielding vehicles have to wait (see section 3.1.3.2). They also contain a set of *Conflict Markers* which mark conflicting streams that must be considered by the yielding vehicles when looking for a gap. Figure 3.24 shows an exemplary priority intersection. Vehicles on the merging *Link* have to wait at the *Priority Rule's* stop line (red) and the *Conflict Marker* on the main *Link* marks the conflicting stream (green line and arrow).
- *Conflict Markers* are placed on *Lanes* and additionally reference the *Link* containing that *Lane*. The Boolean attribute '*allLanes*' indicates whether all *Lanes* of that *Link* or only one are covered by the *Conflict Marker*. Furthermore, a set of *Vehicle Classes* can be defined for which a *Conflict Marker* is valid. Vehicles in the minor stream will not yield to vehicles of the other (not referenced) *Vehicle Classes*.

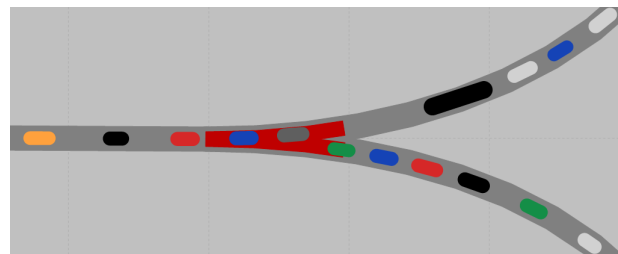




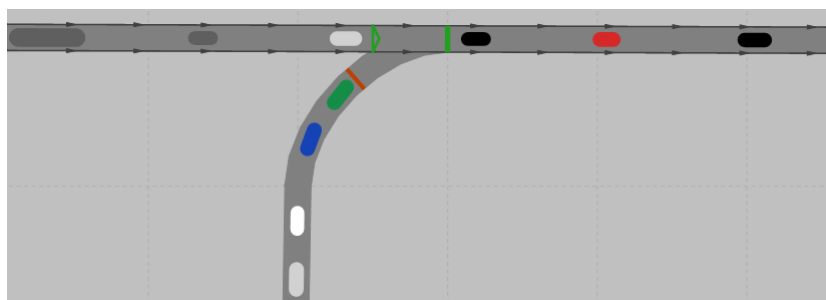
**Figure 3.21:** PTV VISSIM Metamodel: Right of Way at Intersections



**Figure 3.22:** The four statuses of conflict areas in PTV VISSIM. From left to right: passive, two-yields-one, one-yields-two and undetermined.



**Figure 3.23:** An undetermined conflict area (red) between two branching links.



**Figure 3.24:** A priority rule on the merging link (red line) and a conflict marker (green line and arrow) on the main link.

### 3.2.2 Signal Programs

Signal programs are used to control an intersection either to increase its capacity or the safety of the crossing vehicles and pedestrians [30, p. 205f]. If the capacity limit of a priority intersection is reached, minor streams are often neglected and have little opportunities to cross it. Signal-controlled intersections assure that every stream gets a fixed amount of time during which it can cross the intersection.

Before a stream gets the green signal, a certain time is required to avoid accidents with vehicles of conflicting streams which are still leaving the intersection. This time is called intergreen time and can be different for each pair of conflicting streams. The intergreen times of an intersection are usually combined into an intergreen matrix.

The signal heads at an intersection are controlled by a signal program. That program determines the signals of each signal head for each second. Since some streams have multiple signal heads that always display the same signal, signal programs usually aggregate multiple physical signal heads into logical signal groups.

If two signal groups contain non conflicting streams they may receive a green signal at the same time. All signal groups that receive the green signal at the same time are further combined into stages. When the signal program switches from one stage to another there is a short period where no signal group receives green. This time is used to ensure the intergreen times between conflicting streams in consecutive stages.

The duration of a signal program during which all stages receive the green signal exactly once is called cycle time. The duration of each stage's green signal depends on the dominant traffic volume of that stage relative to the dominant traffic volumes of all stages.

Since the dominant traffic volumes can change during a day, the signal program of an intersection can be switched e.g. in the evening or on weekend days. Furthermore, there are signal programs that react to the current traffic on the roads entering the intersection. For instance, induction loops can be used to detect the presence of vehicles and extend or shorten green time of a stage accordingly.

On urban roads, there are often consecutive intersections along the road that are all controlled by traffic lights. It is desirable, that vehicles of one stream passing these intersections do not have to stop at each of the. Therefore, their signal programs can be synchronized. Since signal programs are periodic, their beginning can be changed by an offset. By carefully selecting these offsets, two signal programs can be synchronized such that a stream receives the green signal just when it reaches the second intersection without having to stop. This coordination of signal programs is commonly known as a green wave.

### 3.2.2.1 Signal Programs in SUMO

In contrast to the common notation of signal plans which contain consecutive signals for each signal group, SUMO uses phases to model these signals. Each phase contains the signals for all connections of a junction and specifies for how long they are displayed before switching to the next phase. SUMO also supports traffic light programs that respond to the current traffic which is determined by detectors (see section 3.4.1). Those detectors are implicitly placed on incoming lanes. Furthermore, different traffic light programs can be used at different times of day. The process of switching traffic light programs is performed by a WAUT<sup>1</sup> which contains a list of traffic light programs as well as the time at which they start.

Figure 3.25 shows SUMO's metamodel elements that model traffic light signal programs. In the following, we will list these elements with some of their properties:

- Every *Traffic Light Logic* references the *Junction* it controls and has a *programId*. A *Traffic Light Logic* can also specify a time *offset*. The actual signals, that will be shown by the implicit signal heads of the controlled *Connections*, are modeled as a set of *Phases* which each describe the displayed signals for a certain period of time.
- A *Phase* describes the displayed signals of all controlled *Connections* at a *Junction*. Therefore it contains a list of signals called traffic light *states* - one for each *Connection*. Each *Connection* of a *Junction* has an index (*linkIndex*) that determines the relevant signal in the *Phase*'s list of *states*. By default, these indices are counted clockwise around the controlled intersection with index 0 at '12 o'clock' (north). If a *Traffic Light Logic* controls multiple *Junctions*, the numbering of indices continues in the order of the controlled *Junctions*. Furthermore, signal groups can be modeled by assigning the same *linkIndex* to multiple *Connections* so they always receive the same signal. Every *Phase* has a *duration* for which it is displayed, before the program switches to the next *Phase*. Figure 3.26 shows an exemplary signal program consisting of four *Phases*. During the displayed *Phase* the streams coming from the top and the bottom receive the green signal while the other streams get red.
- The *Traffic Light States* 'red' (*r*), 'amber' (*y*), 'green' (*G*), 'red-amber' (*u*) and 'off' (*O*) are possible signals in a signal program. The signals 'green' and 'off' can be further distinguished into major (*G,O*) and minor (*g,o*) signals. On *Connections* with a 'green major' signal, vehicles do not have to wait for any vehicles in a foe stream to cross the intersection. The 'green minor' signal indicates that vehicles may pass the *Junction* but have to give way to foe streams with a higher priority. Similarly, 'off minor' is a blinking signal which indicates that the signal program is off but vehicles must give way to other streams with the signal 'off major'. The signal 'stop' (*s*) is used to model a green right turn arrow which allows vehicles to drive after they have stopped and observed conflicting streams.

<sup>1</sup>Wochenschaltautomatik (~ weekly switch automatism)

- A *WAUT* allows to use different traffic light programs throughout the day or week. Each *WAUT* has a unique *id* and a program that will be used when the simulation starts (*startProg*). The actual program switches are modeled as a set of *WAUT Switches*.
- *WAUT Switches* take place at a certain *time* during the simulation and switch the current *Traffic Light Logic* to a new one (*to*). The new traffic light program is referenced by its *programId*.
- *WAUT Junctions* couple a *WAUT* with a *Junction* to specify which traffic light should be switched by the *WAUT*.

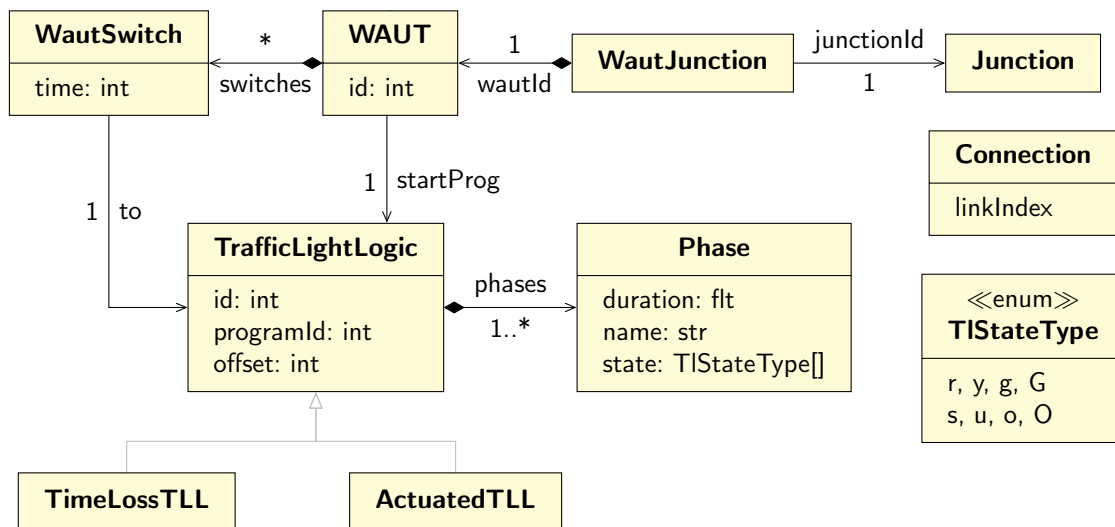
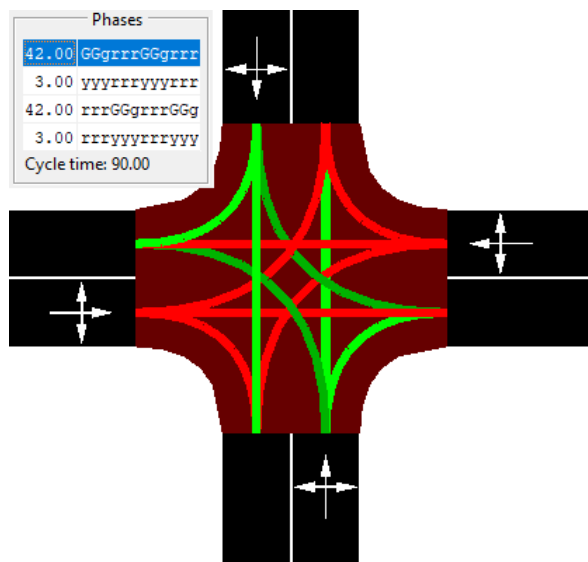


Figure 3.25: SUMO Metamodel: Signal Programs



**Figure 3.26:** A signal program in SUMO consisting of four phases (top left table). The highlighted phase is displayed on the intersection: Each connection is displayed in the color of its signal.

### 3.2.2.2 Signal Programs in PTV Vissim

In PTV VISSIM, intersections can be controlled by signal controllers. Signal controllers execute a signal program to determine which signals should be displayed at a certain time. Signal groups are the smallest logical control unit in such a program. Thus, the signal program has to define the signal of each signal group for every time step. However, one signal group can be represented by multiple signal heads which are the smallest physical control unit in PTV VISSIM. Signal heads merely display the current signal of the signal group they represent. In this way, multiple streams can be controlled by one signal group. Not only signal heads but also stop signs and priority rules can react to the current state of a signal group. PTV VISSIM also provides detectors that can be placed in front of the stop lines of signal heads to adapt the signal program to the current traffic.

Figure 3.27 shows PTV VISSIM's metamodel elements that model traffic lights. In the following, we will list these elements with some of their properties:

- *Signal Controllers* contain the set of *Signal Groups* they control and reference a *Signal Program*. Furthermore, they have a unique number (*no*) and a *name*. To turn off signal heads, the *Signal Controller* that controls them can be deactivated (*active*).
- *Signal Groups* represent aggregated streams that receive the same signal at all times. Additionally, they have a *name* and a unique number (*no*). *Signal Groups* can be assigned different *Signal States* including the basic signals *Red*, *Amber* and *Green*, the combinations *Red-Amber* and *Green-Amber*. Furthermore, each of the three basic signals can be flashing or *Red* and *Green* can be alternating. Finally, the *Signal State* can also be *Off* or *Undefined*.
- Every *Signal Head* references a *Signal Group* (*sg*). They always display the current *Signal State* of their *Signal Group*. In this way, they are indirectly controlled by the *Signal Controller* which determines the signals for every *Signal Group* while *Signal Heads* observe that current *Signal State*. *Signal Heads* can also reference a second *Signal Group* (*orSg*). In that case, they will always show a green signal if at least one of the two *Signal Groups* is green. More precisely, they display the current *Signal State* of the second *Signal Group* as long as the first one is *Red*.
- *Stop Sign* objects can be used to model green arrow signs which allow vehicles to turn right even if other streams on the same *Link* have a red signal. To use a *Stop Sign* in that way, it must reference a *Signal Group*. Every time that *Signal Group* is red, the *Stop Sign* will be activated, which makes vehicles halt at the stop line. They will continue driving as soon as there is a suitable gap in the conflicting stream. As long as the *Signal Group* is green, the *Stop Sign* is deactivated since vehicles turning right do not have to worry about conflicting streams, hence they do not stop and wait for a gap.

- Similar to *Stop Signs*, *Priority Rules* can reference a *Signal Group*. The *Priority Rule* will only be active if the referenced *Signal Group* shows a certain signal (*sigState*). This can be used to model situations where waiting vehicles should not consider vehicles on the major steams behind a red *Signal Head*.
- PTV VISSIM supports *Detectors* that can be placed on *Lanes*. They can reference *Signal Controllers* which in turn can use the measured data from the *Detectors* to adapt the *Signal Program* to the current traffic situation.

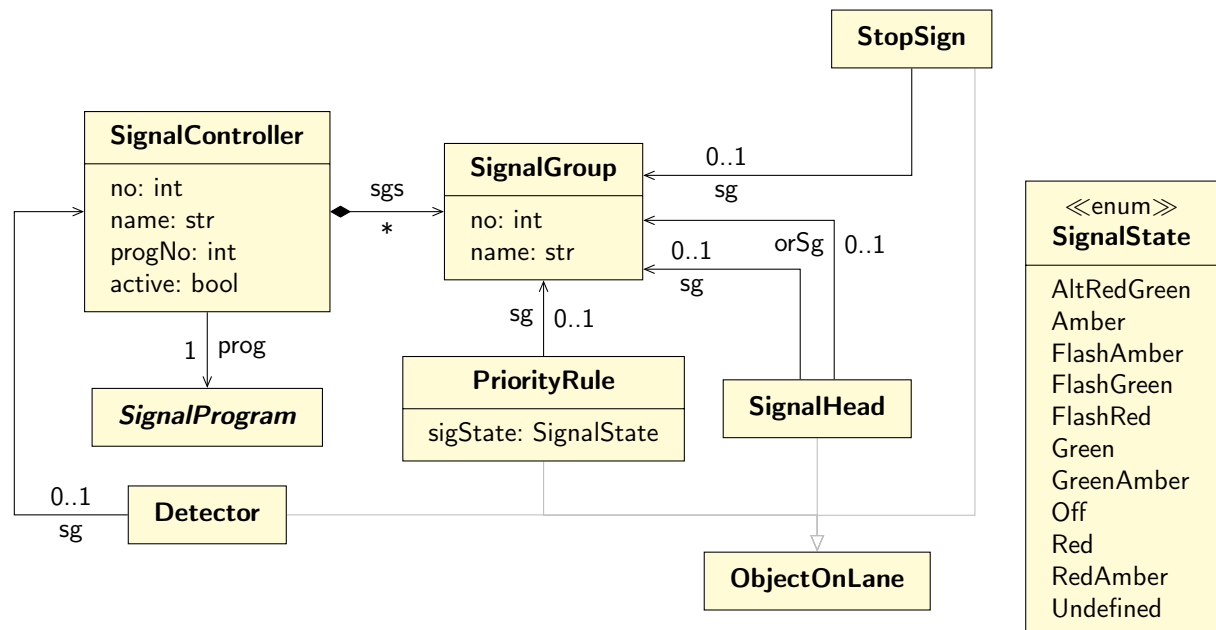


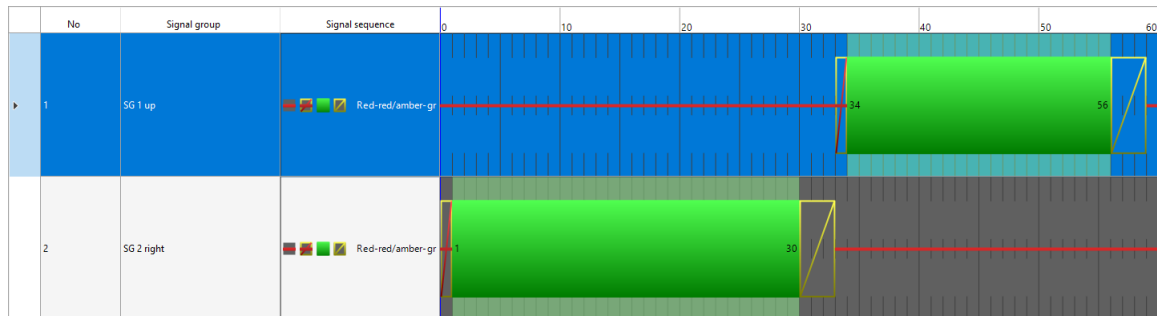
Figure 3.27: PTV VISSIM Metamodel: Signal Controllers

PTV VISSIM supports a wide range of different signal controller types. Some of them conform to real world signal controllers, so the simulation can be coupled with real signal programs. In the following, we will look at the metamodel of fixed-time signal programs (VISIG<sup>1</sup>) as one example of these types.

A fixed-time signal program has fixed cycle time. Once the cycle time is over, the signal program starts anew. Such a program defines the signals for each signal group during this cycle time. Each signal group has a specific sequence of signals. For those signals without a fixed duration, the beginning is defined. Fixed signals specify a duration. The 'free' signals are displayed until shortly before the next free signal begins. Fixed signals are placed between those free signals according to the signal sequence with the specified duration. To check whether a signal program provides enough time for vehicles to clear the intersection before the next signal group receives the green signal, signal programs contain intergreen matrices.

<sup>1</sup>Signal Program Model of PTV VISSIM

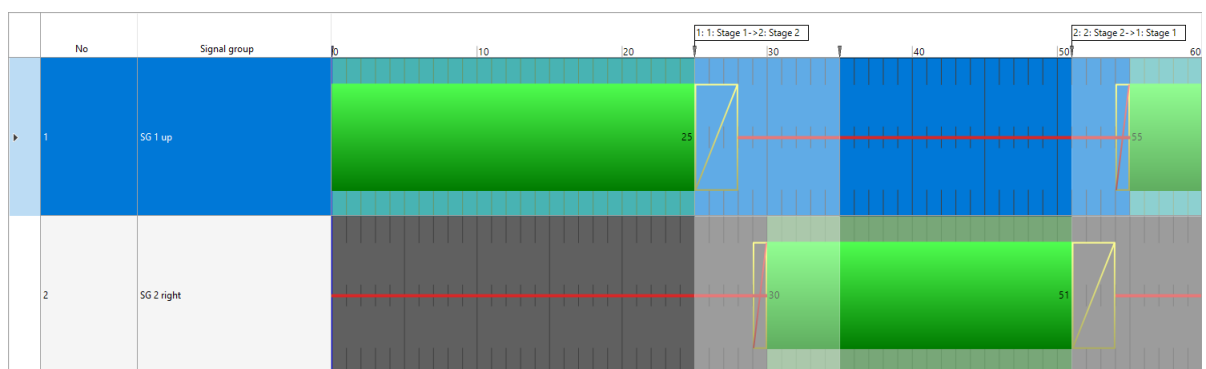
Figure 3.28 shows an exemplary signal program controlling two signal groups. Both have a fixed red-amber time of one second and an amber time of three seconds. The free signals in this example are red and green. Their beginnings are denoted by the labeled bounds of the green bar.



**Figure 3.28:** A signal program controlling two signal groups with a cycle time of 60 seconds.

VISIG allows to group signal groups into stages. All signal groups within one stage receive the green signal simultaneously. All other signal groups get the red signal. With this assignment of signal groups to stages, the sequence of stages within one cycle can be defined. During these stages, the stage assignment defines which signal group gets which signal. However, the transition between stages has to be modeled explicitly to ensure the signal sequence and intergreen times are not violated. Stage-based programs merely specify the starting times of these stage transitions within the cycle. The example in figure 3.29 shows a stage-based signal program with two stages. The stage transitions are grayed out.

Finally, VISIG provides daily programs that can switch between multiple programs at specific times in the simulation. With this, appropriate signal programs can be used for night times or weekend days if there is a different traffic demand at these times.



**Figure 3.29:** A stage-based signal program controlling two stages each containing one signal group. The grayed out areas mark the stage transitions.

Figure 3.30 shows VISIG's metamodel elements that model fixed time signal programs. In the following, we will list these elements with some of their properties:

- Every *Signal Program* has a *name* and an *id* as well as a fixed *cycle time*. For each pair of controlled *Signal Groups*, an intergreen time can be specified in form of an intergreen *Matrix*. *Signal Programs* define the signals for each *Signal Group* along a time axis, in the form of *Signal Program Rows* (*rows*), one for each *Signal Group*.
- Every *Matrix* has an *id* and a *name* and contains a set of *Intergreen entries*.
- *Intergreen* objects represent entries in intergreen *Matrices*. They have a *value* which is the intergreen time between the *clearing* and *entering Signal Group*.
- Each *Signal Program Row* (*SigProgRow*) references the *Signal Group* for which the signals are defined as well as the used *Signal Sequence*. For each fixed signal in that *sequence*, the *Row* contains a *Fixed* signal duration. For each free signal, the *Row* contains that *Free* signal's beginning. Those free signals are displayed until the next free signal starts. If there is at least one fixed signal between them in the *Signal Sequence*, it is added before the beginning of the second free signal.
- A *Signal Sequence* (*SignalSeq*) is a sequence of signal *States* that describes the order in which signals are displayed within one cycle. It also has a *name* and a unique *id*.
- A *State* represents one *signal* within a *Signal Sequence*. It specifies a *default duration* (*defDur*) for which the *signal* has to be displayed. Furthermore, a *State* can represent a *fixed* signal with a fixed *duration* (*defDur*) or an open signal with a minimum *duration* (*defDur*). The Boolean attribute '*closed*' states whether vehicles are allowed to drive when receiving the *signal*.
- *Free* objects represent the beginning of a free *signal* in a *Signal Sequence*. The attribute '*begin*' specifies a point of time within a cycle at which the *signal* (*sig*) begins.
- *Fixed* objects represent *signals* with a fixed duration in a *Signal Sequence*. The attribute '*dur*' specifies the fixed duration of the *signal* (*sig*).
- A *Basic Program* is a *Signal Program* that additionally specifies an *offset* which is a time delta between the start of the simulation and the start of the first complete cycle. This can be used to synchronize *Signal Controllers* of consecutive intersections e.g. to achieve a green wave. Furthermore, *Basic Programs* define a *switch point* which is a certain point of time during the cycle at which the *Signal Program* can be swapped. When switching to a new *Signal Program*, that new program is required to use the same signals for all *Signal Groups* as the former one at its *switch point*.
- *Interstage Programs* are *Signal Programs*, that specify the signals during the transition between the two *Stages from* and *to*. So they do not cover a whole cycle but only a short period within a cycle.
- A *Stage* has an *id* and a *name*. It explicitly specifies which *Signal Groups* receive the green signal (*on*) and which receive the red signal (*off*).



- A *Stage Program* assembles the *Inter Stage Programs* that specify the transition between its *Stages*. It defines the *beginning* of each *Inter Stage Program* during the cycle. The signals between those transitions are derived from the assignment of *Signal Groups* to the *Stages*.
- *Inter Stage Start* objects define the time within a cycle (*beginning*) at which the referenced *Inter Stage Program* starts.
- *Daily Programs* can be used to switch between multiple *Signal Programs*. The program switches are modeled as a set of *Program Starts*.
- *Program Starts* reference a *Signal Program* and specify a *time* during the simulation at which the referenced program starts.

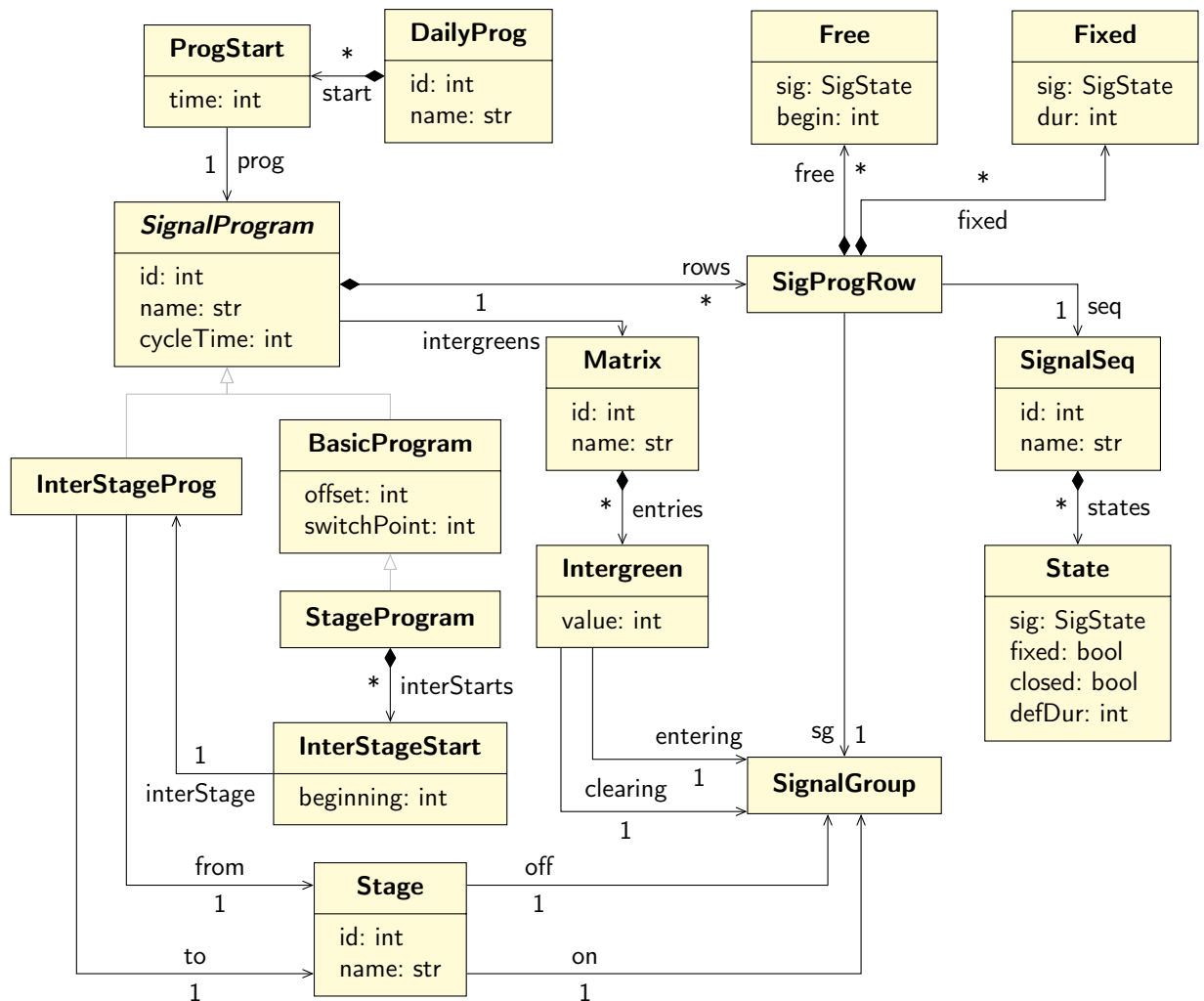


Figure 3.30: VisSIG Metamodel: Signal Programs

## 3.3 Modeling Traffic

In section 3.1 we discussed how the road infrastructure is represented in PTV VISSIM and SUMO. In section 3.2, we looked at their representations of traffic control rules like signal programs and priority rules. With these two prerequisites, we can now look at the behavior of vehicles when they are using the infrastructure and react to the installed traffic rules. The behavior of a vehicle depends on the vehicle's static properties (e.g. the power of its engine or its length) as well as the driver's behavior which we will look at in sections 3.3.1 and 3.3.2. Usually, a vehicle and its driver are modeled as one entity, which is why we will use the expression 'the vehicle's driving behavior' instead of 'the driver's behavior'. After describing the behavior of a single vehicle, we still have to specify how many vehicles travel from one point to another. Vehicle demand and vehicle routes are described in section 3.3.3.

### 3.3.1 Vehicle Properties

Every vehicle has properties that influence its behavior on the road that are independent of the driver. This includes properties like a vehicle's length or the engine's power which defines a vehicle's maximum possible speed. Those static properties do not change during the simulation and are often shared by multiple vehicles. Hence, all static properties can be enclosed in vehicle types which can then be referenced by the individual vehicles. For example, multiple sports cars can reference the same vehicle type *SportsCar* while all family vans reference the same vehicle type *FamilyVan*. In the following sections, we will see how PTV VISSIM and SUMO represent these static vehicle properties.

#### 3.3.1.1 Vehicle Properties in SUMO

In SUMO, every vehicle that is created during the simulation has certain dynamic properties like its current speed or acceleration as well as some static properties like its length and width. These properties are modeled as attributes of vehicle types. In SUMO, vehicle types are combined into more abstract, distinct vehicle classes. The two vehicle types *SportsCar* and *FamilyVan* from the introduction's example both describe passenger cars, hence they both belong to the vehicle class *passenger*. Other vehicle classes are *pedestrian*, *bicycle* or *tram*. In SUMO, these vehicle classes are used to model permissions on certain sections of the network for different kinds of vehicles (see section 3.1.1.1). In this way, not every vehicle type that models a passenger car has to be disallowed from bicycle paths. Instead, all passenger cars can be disallowed from a lane simply by referencing a single vehicle class.

Figure 3.31 shows SUMO's metamodel elements that model static properties of vehicles. In the following, we will list these elements with some of their properties:

- *Abstract Vehicle Type* is the abstract supertype of concrete *Vehicle Types* and *Vehicle Type Distributions*. Every *Abstract Vehicle Type* has a unique *id* and a *probability* that is used when drawing *Vehicle Types* from a distribution.
- *Vehicle Types* specify the static properties of a certain kind of vehicle. Every *Vehicle Type* has a unique *id* and belongs to one *Vehicle Class*. A vehicle's *length*, *width* and *color* are defined as well as the number of people (*personCapacity*) it can transport. It also specifies the time it takes for a person to enter the vehicle (*boardingDuration*). Finally, every *Vehicle Type* has a *maximum speed*, a maximum *acceleration* and a maximum *deceleration* (*emergencyDecel*). Those values are the limits that can be physically achieved by a vehicle. Whether the driver actually wants to use the maximum speed is an aspect of the vehicles driving behavior which is discussed in section 3.3.2.
- As an alternative to referencing one specific *Vehicle Type*, *Vehicle Type Distributions* can be referenced. In that case, a concrete *Vehicle Type* is drawn from a *Distribution* when a vehicle is created. Every *Abstract Vehicle Type* can be assigned a *probability*. This probability is used to determine the likelihood of each *Vehicle Type* to be drawn from the distribution. Since *Vehicle Types* can occur in multiple *Distributions*, their *probabilities* will be scaled for each *Distribution* individually. Thus, it is not necessary that the sum of a distribution's *probabilities* equals one.
- *Vehicle Classes* are categories, that can be used to group *Vehicle Types*. These classes are used to restrict certain *Lanes* to a specific kind of vehicle: e.g. bicycle paths which only vehicles of the *Vehicle Class* *bicycle* are allowed to use. The most important *Vehicle Classes* are *pedestrian* to model people on sidewalks, *passenger* to model private cars, *bicycle*, *bus* and *tram*.

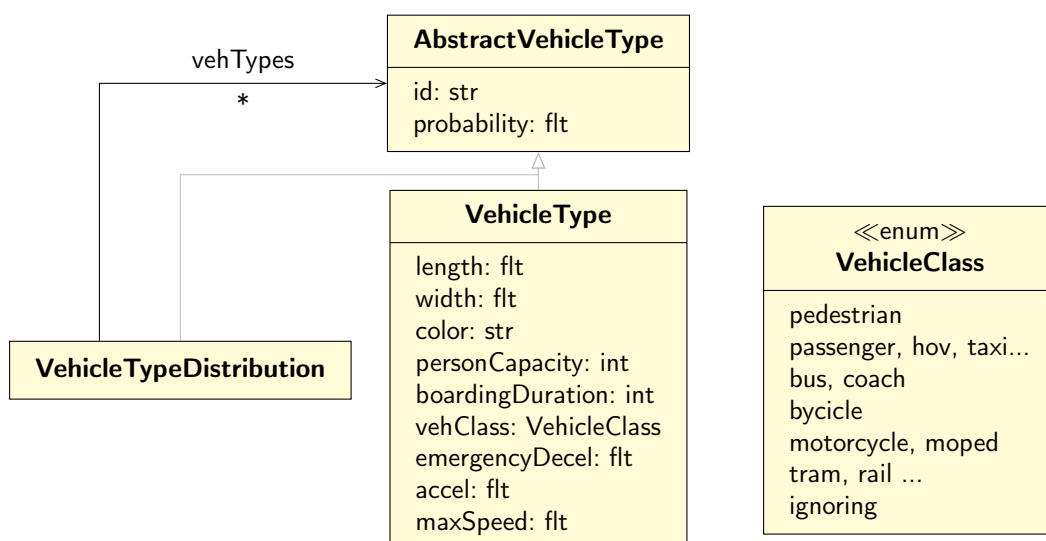


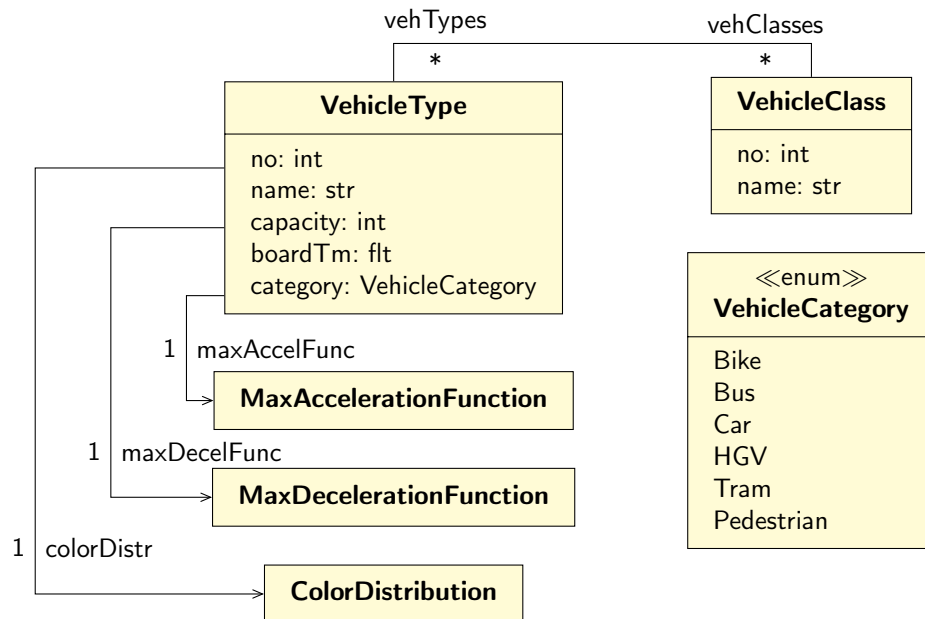
Figure 3.31: SUMO Metamodel: Vehicle Properties

### 3.3.1.2 Vehicle Properties in PTV Vissim

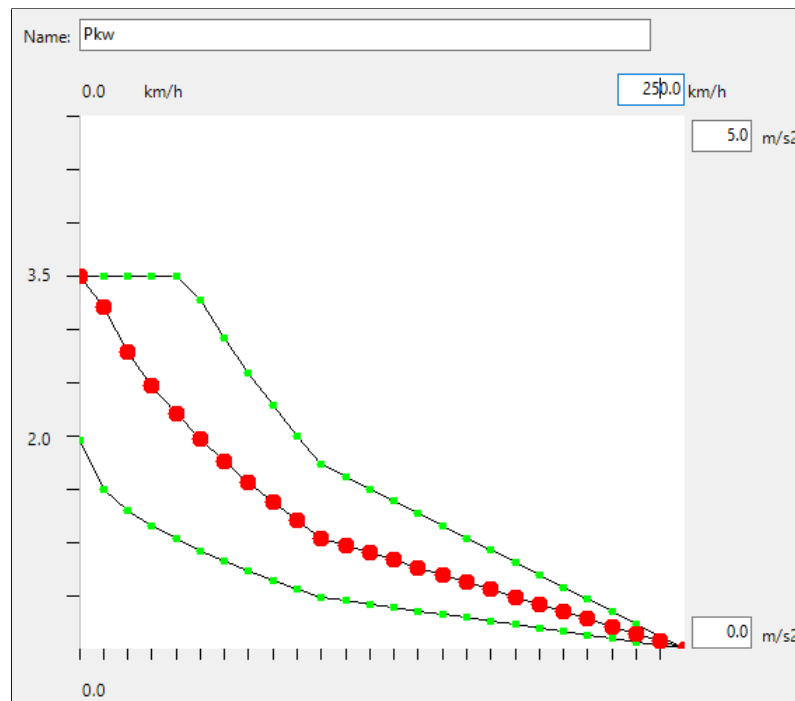
PTV VISSIM models static vehicle properties like the vehicle's color, its passenger capacity or the maximum possible deceleration as attributes of vehicle types. Some properties of vehicle types are based on distributions, in which case instantiating vehicles receive one value drawn from the distribution. Furthermore, each of these vehicle types belongs to exactly one vehicle category which specifies the basic behavior. For example, vehicle types of the category *Tram* are not allowed to change lanes and pedestrians follow different behavior models than cars. While vehicle categories are disjoint groupings predefined by PTV VISSIM, vehicle types can also be grouped into - not necessarily disjoint - custom vehicle classes.

Figure 3.32 shows PTV VISSIM's metamodel elements that model static properties of vehicles. In the following, we will list these elements with some of their properties:

- Every *Vehicle Type* has a unique number (*no*) and a *name* and belongs to exactly one predefined *Vehicle Category*. Aside from these *Vehicle Categories*, *Vehicle Types* can be part of multiple *Vehicle Classes*. *Vehicle Types* reference a *Maximum Acceleration Function* which describes the maximum acceleration a vehicle can achieve. The acceleration abilities of a vehicle depend on its current speed, thus the maximum acceleration is a function of speed. Furthermore, the acceleration function can be used to model the maximum speed  $v_{max}$  a vehicle can reach by setting the acceleration to zero for all velocities greater than  $v_{max}$ . Given a certain speed, the *Maximum Acceleration Function* describes a range of acceleration values as well as a median. In this way, the maximum acceleration of an instantiating vehicle can be derived from this distribution of acceleration values using linear interpolation between the median and the lower or upper bound. The *Maximum Deceleration Function* is used accordingly. Figure 3.33 shows an exemplary *Maximum Acceleration Function* with median values (red) as well as a minimum and a maximum bound (green). Public transport *Vehicle Types* can define a maximum passenger *capacity* and the *boarding time*.
- A *Vehicle Type's* *Vehicle Category* determines its basic behavior. The default category is *Car*. All other *Vehicle Categories* are based on *Car* with some differing properties. For example, the category *HGV* influences the acceleration and lane changing behavior of heavy vehicles, *Trams* are not allowed to change lanes at all and *Bikes*, *Pedestrians* and *HGV* can only be occupied by one person.
- *Vehicle Classes* are custom groupings of *Vehicle Types*. They are used to treat multiple *Vehicle Types* equally e.g. when assigning new desired speeds at *Desired Speed Decisions* (see section 3.1.2.2). Each *Vehicle Type* can belong to multiple *Vehicle Classes*, so they do not have to be disjoint. *Vehicle Classes* have a *name* and a unique number (*no*).



**Figure 3.32:** PTV VISSIM Metamodel: Vehicle Properties



**Figure 3.33:** An exemplary acceleration function describing acceleration values over speed. The function describes the acceleration as a series of data points (red dots). The function's values between the data points are derived by linear interpolation. The function also describes a speed dependent minimum and maximum bound to the acceleration values (green dots). Vehicles receive an acceleration function that is interpolated between the median values and the minimum or maximum bound.

### 3.3.2 Driving Behavior

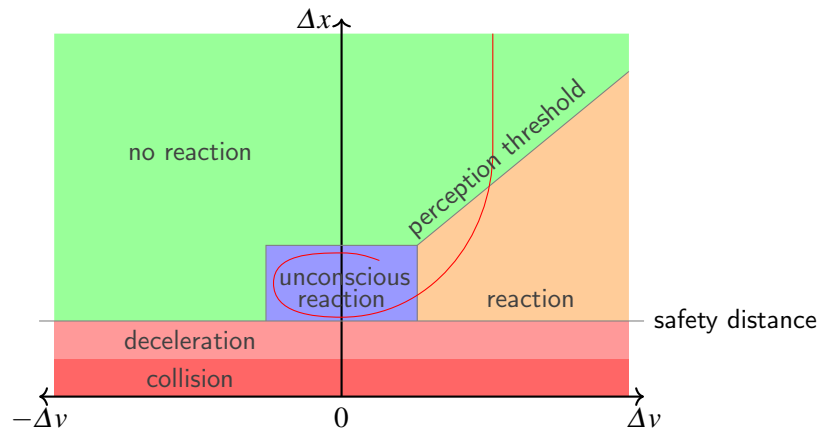
Modeling the driving behavior of vehicles is an essential part of microscopic traffic simulation since each vehicle is simulated individually. For example, the way how following vehicles react when the leading vehicle decelerates, can influence the creation of traffic jams. In the history of driving behavior models, different approaches have been developed concerning different aspects like the behavior while following a leading vehicle, changing lanes or lateral movement within a lane. The following models are dynamic models describing vehicle movements. Their static representation in the SUMO and PTV VISSIM metamodel will be discussed in sections 3.3.2.1 and 3.3.2.2.

SUMO and PTV VISSIM both support different driving behavior models. As car following models they both support the models by Wiedemann from 1974 [35] and 1999 (cf. [36]). More precisely, PTV VISSIM only supports Wiedemann '74 and Wiedemann '99. Since we want to transform SUMO models into PTV VISSIM models, we will exemplarily describe the Wiedemann '99 model.

The Wiedemann models are psycho-physical car following models as they model the behavior of drivers based on their perception (psychological) of their environment's behavior (physical). The core idea is that drivers perceive the relative velocity between themselves and the leading vehicle. Additionally, perception thresholds are considered, below which the driver does not perceive the relative velocity and will not react to the leading vehicle. For example, if the leading vehicle is slower but still far away, the following vehicle cannot perceive the speed difference. Only when it approaches the leading vehicle, at some point the driver will cross the perception threshold and start to respond to the leading vehicle. The process of approaching and following a vehicle is divided into four interaction states as shown in figure 3.34:

- free-driving (green): the following vehicle does not react to the leading vehicles behavior
- approaching (orange): the following vehicle does perceive and react to the leading vehicles behavior
- following (blue): the following vehicle closely follows the leading vehicle and reacts unconsciously
- breaking (red): the following vehicle brakes as it is closer to the leading vehicle than the desired safety distance

Figure 3.34 also contains a red curve, which displays an exemplary following vehicle. If a vehicle is in the free-driving state it is not influenced by any leading vehicle and uses its desired speed or accelerates to reach it. Once the response threshold is reached, the driver will decelerate. If the distance to the leading vehicle and the relative velocity are small enough, the following state is reached. In that state, the following vehicle drives with a similar speed as the leading vehicle while trying to keep the safety distance. If the distance to the leading vehicle falls below the safety distance, the driver will brake to avoid a collision.



**Figure 3.34:** Distance over relative velocity with interaction states and an exemplary trajectory of an approaching vehicle (red curve) based on [36, fig. 1]

In the Wiedemann '99 model, the thresholds between these states and the acceleration properties of the following vehicle are based on a set of ten parameters [1, p. 297]:  $cc0$  -  $cc9$ .

In contrast to the psycho-physical Wiedemann models in PTV VISSIM, SUMO's Krauss model is a safety distance model. It assumes that a following vehicle tries to drive with its desired speed while keeping a minimum safety distance between itself and the leading vehicle. With this, the following vehicle wants to assure that it can break without colliding with the leading vehicle in case the latter decelerates.

Estimating distances is difficult for drivers [30, p.65], which is why the headway is given as a time gap. In this way, it is also independent of the vehicles speed. With increasing speed, a longer distance can be traveled in the same time. Thus, the safety distance increases as the vehicles speed increases. A common rule of thumb is to keep a time gap of two seconds.

Aside from car following models, there are also models that describe other driving behavior aspects like lane changing models. However, SUMO and PTV VISSIM do not support the same models. In contrast to network elements, which have different representations in SUMO and PTV VISSIM but the same behavior, driving behavior models are algorithms for computing the movements of vehicles. These algorithms in PTV VISSIM and SUMO have different behaviors, which makes it hard to translate the parameter setting of one algorithm to the other. This is why we will only consider the parameter settings of the Wiedemann '99 car following model and leave out the other driving behavior models.

### 3.3.2.1 Driving Behavior in SUMO

The behavior of vehicles in the simulation depends on the properties of a vehicle like its length or emergency deceleration on the one hand. On the other hand, a vehicle's driver has a big influence on its behavior, especially when interacting with other vehicles. In SUMO, driver and vehicle are modeled as one unit. Similar to the static properties of vehicles, its driving behavior is also modeled in the *Vehicle Type* class. The properties of the driving behavior are grouped into car following properties, lane changing properties and junction behavior properties. Aside from these three groups of behavior properties, there are some general properties concerning the drivers desired speed or deceleration that are relevant in all three of them.

Figure 3.35 shows SUMO's metamodel elements that model general driving behavior properties. In the following, we will list these elements with some of their properties:

- *Vehicle Types* contain both a set of static vehicle properties (see section 3.3.1.1) and a set driving behavior properties. The latter includes the drivers desired deceleration (*decel*) and a *speed factor*. To model vehicles that drive a little faster or slower than the speed limit, vehicles have a *speed factor* that is applied to all speed limits. As long as it does not surpass the vehicle's maximum speed, this scaled speed limit will be used as its current desired speed. Not all vehicles react to speed limits equally, hence the *speed factor* is modeled as a *Normal Distribution*. Furthermore, each *Vehicle Type* references a *Car Following Model* which models the vehicle's longitudinal behavior, a *Lane Changing Model* which models its lateral behavior and a *Junction Behavior Model* to describe how vehicles cross intersections.
- When vehicles are created during the simulation, their *speed factor* will be drawn from a *Normal Distribution*. In this way, the simulation will contain some careful and some aggressive vehicles. The distribution can be customized by specifying the *mean speed factor* and the standard deviation (*dev*). Additionally, a *minimum* and *maximum speed factor* can be defined to avoid extremely low or high speeds.

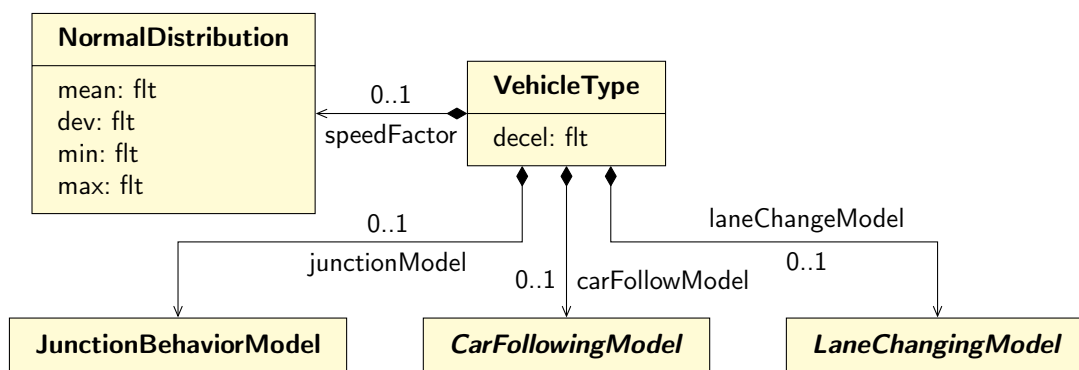
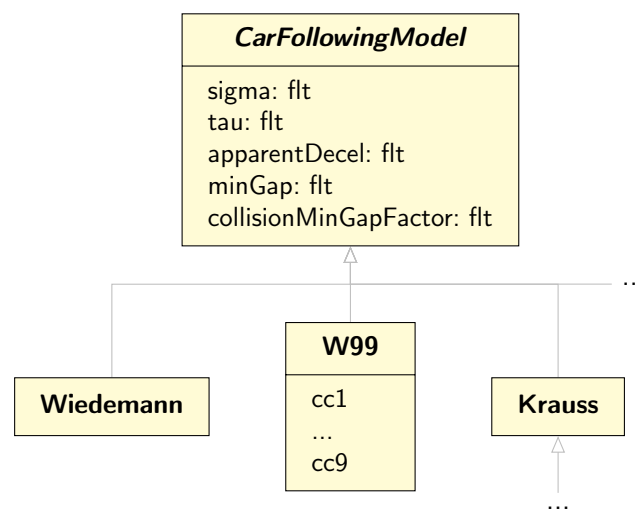


Figure 3.35: SUMO Metamodel: Driving Behavior



One of the advantages of SUMO being open source is that SUMO can be easily extended by new driving behavior models. This is why SUMO supports a large set of Car Following Models. The standard Car Following Model in SUMO is based on the Krauss Model [39]. Furthermore, SUMO supports some variations of the Krauss model as well as the two car following models by Wiedemann from 1974 (Wiedemann, [35]) and 1999 (W99, cf. [36]) which are also supported by PTV VISSIM. SUMO supports more car following models than those three shown in figure 3.36. However, we will only consider the Wiedemann '99 model in this thesis.

Figure 3.35 shows SUMO's car following models. In the following, we will list the general properties as well as the Wiedemann 99 model:



**Figure 3.36:** SUMO Metamodel: Car Following Models

- *Car Following Model* is the abstract supertype of all concrete *Car Following Models* in SUMO. It holds those attributes, that all concrete models have in common. *Sigma* is a value in  $[0,1]$  that models driver imperfection. In the *Krauss* model, if *sigma* is not zero, a vehicle's speed will randomly fluctuate. Other models use *sigma* similarly. *Tau* models a vehicle's desired time headway which they want to maintain as a safety time gap in case the leading car breaks. The expected maximum deceleration of the leading car is given in *apparentDecel*. If the gap between two vehicles falls below *minGap* - the minimum distance between two standing vehicles - SUMO registers a collision. This threshold for collisions can be scaled with the *collisionMinGapFactor*.
- The *Wiedemann 99* model uses the parameters *cc1* - *cc9*. Only the parameter *cc0* is not contained in SUMO's *Wiedemann 99* model since the stand still distance is already covered by the attribute *minGap* that all *Car Following Models* share.

SUMO supports three different lane changing models and a junction behavior model. However, as described in the introduction above, these models are not supported in PTV VISSIM which is why we will not look at the model parameters in detail.

### 3.3.2.2 Driving Behavior in PTV Vissim

In section 3.3.1.2 we discussed PTV VISSIM's metamodel elements, that describe static properties of vehicles. In this section, we will look at driving behavior properties of vehicles like the way they follow a leading vehicle, switch lanes or react to an amber signal at a crossing. PTV VISSIM models driver and vehicle as a single unit, which is why some driving behavior properties like the desired acceleration are modeled in the *Vehicle Type* class together with the vehicle's static properties. However, the car following behavior, the lane changing behavior, the lateral behavior within one lane and the behavior at signal-controlled crossings are contained in the *Driving Behavior* class.

Figure 3.37 shows PTV VISSIM's metamodel elements that model driving behaviors. In the following, we will list these elements with some of their properties:

- *Vehicle Types* hold a *Desired Acceleration Function* and a *Desired Deceleration Function*. Both are functions of velocity similar to the *Vehicle Type's Maximum Acceleration/Deceleration Function* (see section 3.3.1.2). Vehicles in the simulation use their *desired acceleration* as long as it does not surpass their maximum acceleration. The *desired deceleration* is used accordingly.
- Most of a vehicle's driving behavior is captured in the *Driving Behavior* class. Every *Driving Behavior* has a unique number (*no*) and a *name*. It describes the desired *standstill distance* at stop lines and the *increased acceleration (IncrsAccel)* when a leading vehicle accelerates. The Boolean attribute '*EnforceAbsBrakDist*' states whether a vehicle keeps the absolute safety distance. This means they are able to stop without a collision even if the leading vehicle stops without a braking distance. Besides these basic properties, *Driving Behaviors* additionally consist of a *Car Following Model*, a *Lane Changing Model*, a *Lateral Behavior* and a *Signal Behavior*. Some of the *Car Following* and *Lane Changing* parameters can be assigned multiple values depending on the leading vehicle's *Vehicle Class* (*VehClassFollowBehav* and *VehClassLatBehav*).
- *Wiedemann99* is a *Car Following Model*. It holds the parameters *cc0* and *cc2* up to *cc9*. In PTV VISSIM, the *cc1* parameter is modeled as a *Time Distribution*. So the additive time gap of each vehicle's safety distance is drawn from a distribution.

Figure 3.37 shows that vehicle types do not reference a driving behavior. This is because PTV VISSIM does not assign each vehicle one driving behavior upon their creation. Instead, vehicles receive a new driving behavior every time they enter a new link. The reason for separating the driving behavior from the vehicle type is that the behavior of vehicles depends on the road they drive on. For example, vehicles behave differently on motorways than they do on urban roads. Therefore, each link is assigned a link behavior which in turn assigns each vehicle class a driving behavior. Aside from the behavior on links, a vehicle's behavior at conflict areas or stop lines depends on the intersections themselves rather than the individual driver. This is why those traffic control elements also hold some behavior properties.

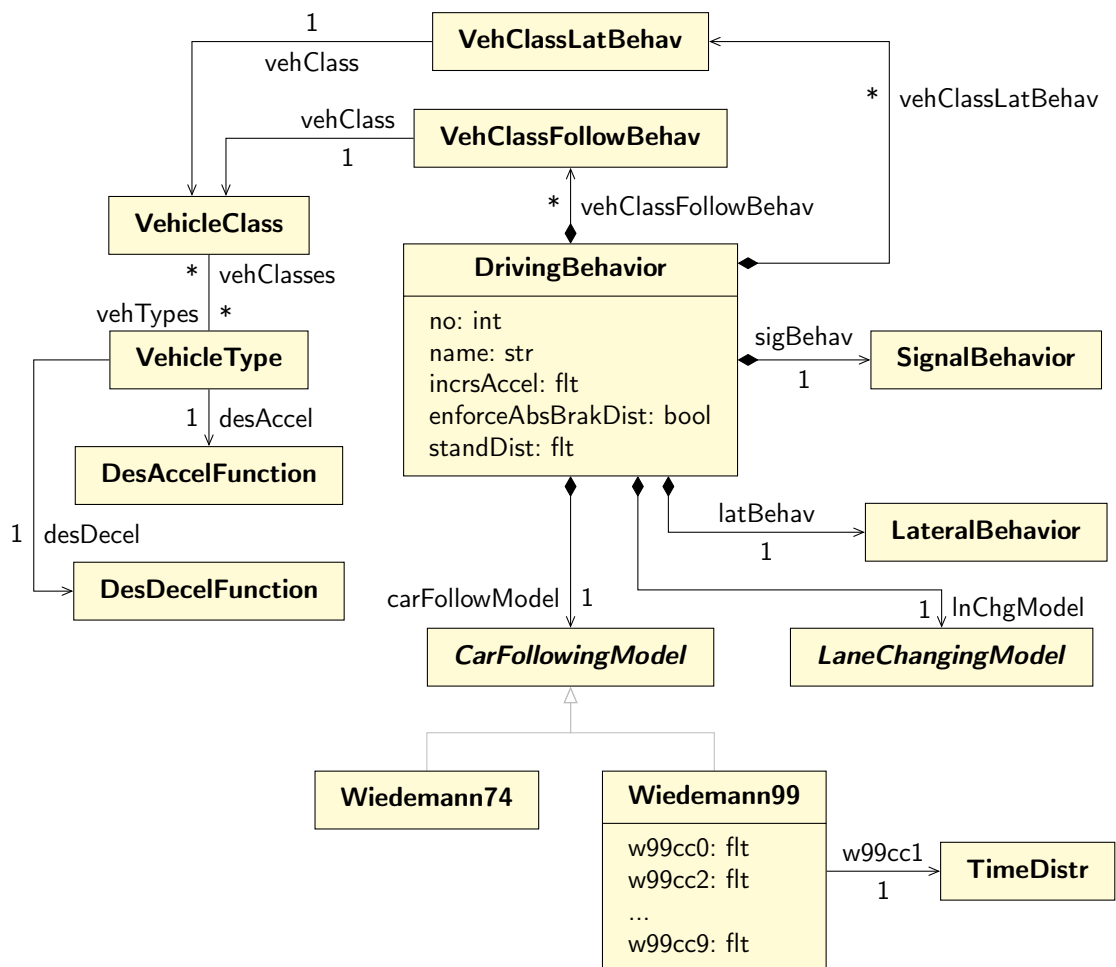


Figure 3.37: PTV VISSIM Metamodel: Driving Behavior

Figure 3.37 shows PTV VISSIM's metamodel elements that model these infrastructure dependent driving behavior properties. In the following, we will list these elements with some of their properties:

- *Link Behavior Types* have a *name* and a unique number (*no*). They assign *Driving Behaviors* to different *Vehicle Classes* using '*VehClassDrivingBehavior*' objects, which link one *Vehicle Class* to exactly one *Driving Behavior*. In this way, when a vehicle of a certain *Vehicle Class* enters a *Link*, the vehicles new *Driving Behavior* can be determined.
- *Connectors* are *Links* that allow vehicles to leave one lane and enter another. So they enable vehicles to pursue routes that include more than just one *Link*. In some situations, vehicles have to change the lane in order to reach a *Connector* that belongs to their route. The time at which vehicles start to switch to an appropriate lane depends on the distance at which they are aware of the *Connector*. This *lane change distance* highly influences the lane changing behavior of vehicles especially when it comes to routing decisions (see section 3.3.3.2).
- *Conflict Markers* define a *minimum headway* which is the minimum distance behind the *Conflict Marker*, that has to be free before a waiting vehicle will continue to drive. They also define a *Minimum Gap Time* which is the minimum time gap between the *Conflict Marker* and the next approaching vehicle. This time gap is the time an approaching vehicle needs to reach the *Conflict Marker*. However, vehicles with a speed larger than '*maxSpeed*' are not considered when looking for a gap.
- *Conflict Areas* can specify whether vehicles in the minor stream *avoid blocking major* streams. On the other hand, a percentage of vehicles in the major stream that *avoid blocking minor* streams can be defined. Furthermore, the percentage of vehicles in the minor stream that *anticipate the routes* of vehicles in the major streams can be specified. If for example an approaching vehicle indicates to leave the main road by turning right before reaching the *Conflict Area*, some vehicles anticipate that route and start driving. Which gaps a waiting vehicle will accept is defined per *Vehicle Class* in the form *Vehicle Class Gaps*.
- A *Vehicle Class Gap* defines for one *Vehicle Class* which gaps its vehicles will accept at a *Conflict Area*. It specifies a *front gap* which is the time gap between a vehicle in the major stream leaving and a vehicle from the minor stream entering the *Conflict Area*. A *rear gap* is specified accordingly as the time gap between a minor-stream-vehicle leaving and a major-stream-vehicle entering the *Conflict Area*. Furthermore, a *safety distance factor* can be defined that is multiplied with a vehicle's safety distance.
- *Signal Heads* can define a *compliance rate*. A rate smaller than 100 % results in vehicles ignoring the *Signal Head*.

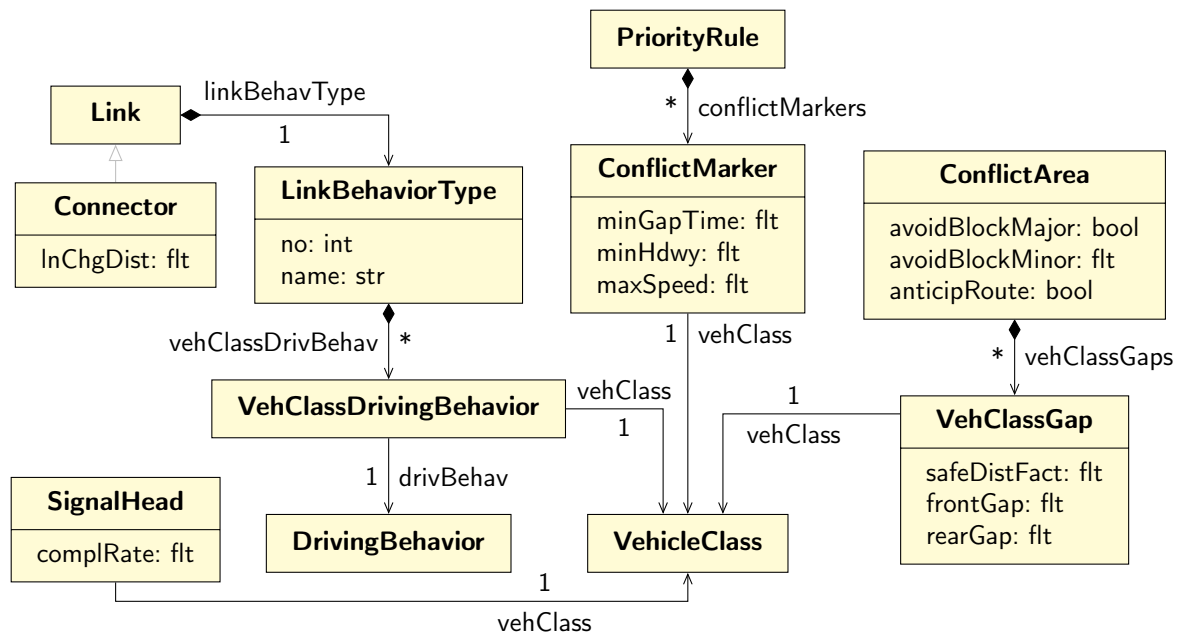


Figure 3.38: PTV VISSIM Metamodel: Link dependent Driving Behavior

### 3.3.3 Traffic Demand

In section 3.3.2 we established how the driving behavior of single vehicles is modeled in PTV VISSIM and SUMO. These driving behavior models describe how drivers react to other vehicles. Especially lane changing models require the vehicles to know their route so the correct lane can be selected in case a road splits. In this section, we will show how multiple vehicles are inserted into the network and how they can be assigned individual routes.

#### 3.3.3.1 Traffic Demand in SUMO

In SUMO, vehicles are assigned a route as soon as they are created. This means that vehicle movements are defined with a large scope reaching from the vehicle's origin to its destination. Vehicles will move through the network along a predefined route which consists of a sequence of connected edges. Once a vehicle has reached its destination, it is removed from the network.

Usually, there are multiple routes that lead from the origin to the destination. Since vehicles tend to distribute over all available routes, those routes have to be modeled separately. To model the distribution of vehicles to different routes, each route can be assigned a probability. In addition to complete routes, SUMO supports incomplete routes called trips. A trip only defines an origin and destination edge but leaves out the intermediate edges. The actual route is computed during the simulation as the fastest path at the time, the vehicle is created.

Aside from the vehicle's routes, SUMO allows to model the number of vehicles that are inserted into the network. Therefore, SUMO provides different options concerning the points in time at which vehicles are inserted into the network. When vehicles are created, they are assigned a vehicle type which is either a constant type or it is drawn from a distribution. Every vehicle type in such a distribution has a certain probability.

Figure 3.39 shows SUMO's metamodel elements that model traffic demand and vehicle routes. In the following, we will list these elements with some of their properties:

- Every *Movement* through the network, whether it is an exact vehicle route or an incomplete trip, has a point where it starts to travel through the network and a point where it leaves the network. Some properties about the departure and arrival of a vehicle are modeled as *Departure State* and *Arrival State*. Additionally, every *Movement* has a unique *id*, an initial number of persons (*personNumber*) and a *departure* time. To determine the *Vehicle Types* of the vehicles performing a *Movement*, either a concrete *Vehicle Type* or a *Vehicle Type Distribution* is referenced (see section 3.3.1.1).
- *Vehicle Movements* are planned *Movements* of a single vehicle along a complete *Route*. That *Route* is either directly referenced or drawn from a *Route Distribution*.
- *Abstract Route* is the abstract supertype of *Routes* and *Route Distributions*. Every *Abstract Route* has a unique *id* and a *probability* that is used when drawing *Routes* from *Route Distributions*.
- A *Route* is a complete path through the network consisting of a list of *edges*. Here, 'complete' means that every two consecutive edges in that list must be connected.
- A *Route Distribution* references a set of *Routes*, each with an individual *probability*. The sum of these *probabilities* does not have to equal one. Instead the *probabilities* are scaled before a *Route* is drawn from the *Distribution*. A *Route Distribution* can also reference another *Route Distribution* with a *probability*.
- As an alternative to specifying *Movements* as *Vehicle Movements* with a complete *Route*, incomplete routes called *Trips* can be used. A *Trip* describes the *Movement* of a single vehicle, but instead of a *Route*, which consists of a list of connected *Edges*, only the origin (*from*) and the destination *Edge* (*to*) are referenced. The actual path of a vehicle is determined during the simulation as the fastest path between those two *Edges*.
- While *Trips* and *Vehicles* describe *Movements* of a single vehicle that departs at a certain time, *Flows* can be used to define repeated vehicle movements. Therefore, each *Flow* references a *Movement* whose departure time is omitted. Instead, *Flows* define a time interval [*begin*, *end*] during which vehicles will be inserted into the network to perform the referenced *Movement*. When and how many vehicles are inserted depends on the concrete *Flow* type.
- A *Number Flow* is a *Flow* that inserts a given *number* of vehicles into the network equally spread over the defined interval [*begin*, *end*].

- Instead of *Number Flows* which specify the total *number* of vehicles over the *Flow*'s complete time span, *VehPerHour Flows* can be used. These *VehPerHour Flows* are useful if the number of vehicles per hour is known. During the simulation, the vehicles are inserted equally spaced over each hour within the *Flows* time span.
- *Period Flows* are *Flows* that define a time period between vehicle insertions. In this way, every *period* seconds a vehicle is inserted.
- A *Probability Flow* specifies the *probability* for a vehicle to appear. This means that in every simulation second a vehicle can be inserted with the specified *probability*.

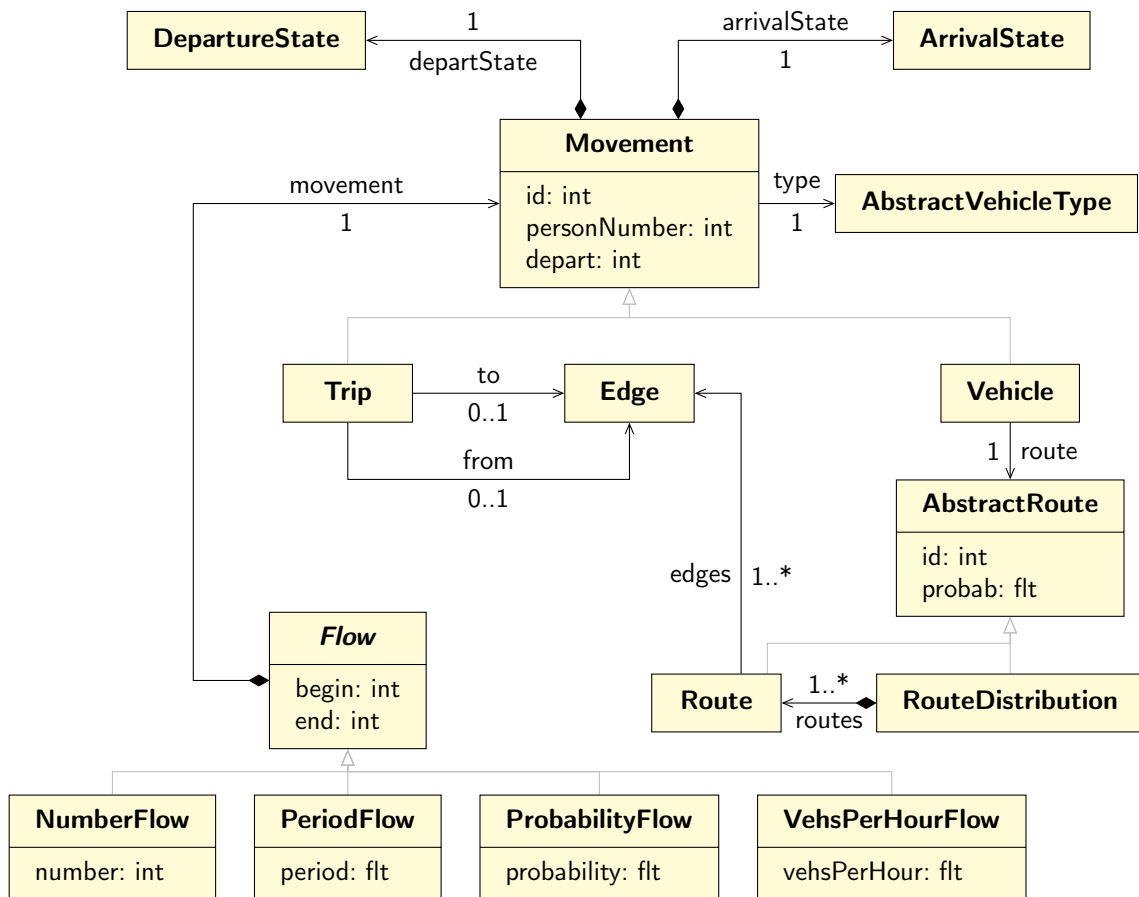


Figure 3.39: SUMO Metamodel: Traffic Demand

SUMO allows vehicles to interrupt their trips to stop at specific locations within the network. This is used to model public transport vehicles that serve different bus stops or private transport vehicles that stop at parking lots.

Figure 3.40 shows SUMO's metamodel elements that model stops. In the following, we will list these elements with some of their properties:

- *Stops* are planned interruptions of vehicle movements. They always take place at a certain *Stop Infrastructure* like a bus stop or a parking area (see section 3.1.4.1). Vehicles will interrupt their movement only for a certain period of time which is either given as a simulation time step (*until*) or a minimum stopping *duration*. If both are defined, the vehicle will always adhere to the minimum stopping *duration* and then wait *until* the given time step is reached. If people are boarding e.g. a public transport vehicle, the stopping *duration* can be further exceeded by a maximum *extension* time. If a public transport vehicle reaches a stop, its *Line* can be renamed in case the current *Line* ends at that stop. The Boolean attribute *parking* states whether the vehicle will park at the side of the road or on it which can block following vehicles. *Stops* can either be child objects of a *Movement* or a *Route*. A *Stop's* *index* denotes its position in such a list of *Stop* child objects.

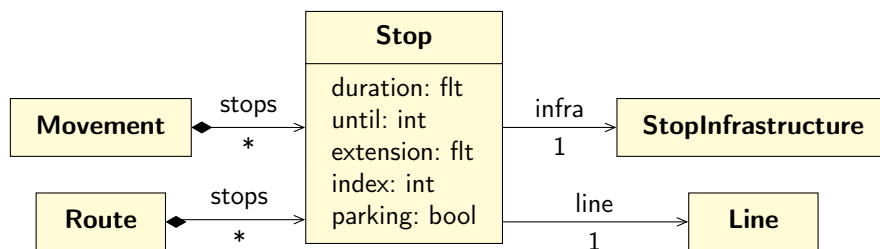


Figure 3.40: SUMO Metamodel: Vehicle Stops

Additionally to vehicle routes and trips, which are assigned to the vehicles during their creation, SUMO supports re-routers which are placed on edges in the network and assign new routes passing vehicles. There are multiple options, how vehicles can be rerouted. We will look closer at rerouting to parking lots, to a new destination and along a new route.



Figure 3.41: A rerouter which assigns a new route to 50 % of the passing vehicles.



Figure 3.42 shows SUMO's metamodel elements that model rerouting. In the following, we will list these elements with some of their properties:

- Every *Rerouter* controls at least one *Edge*. Every vehicle entering such a controlled *Edge* is being rerouted. If not all vehicles are supposed to be rerouted, a *probability* for a single vehicle to be rerouted can be defined. Rerouting can also be restricted to a specific set of *Vehicle Types*. *Rerouters* can also specify a *time threshold*, which is the minimum aggregated waiting time before the *Rerouter* starts to assign new routes to vehicles. The rerouting *Actions* can change for different times of day. Therefore, *Rerouters* can specify multiple time *Intervals* which in turn contain different rerouting *Actions*. Figure 3.41 shows an exemplary rerouter which assigns new routes to 50 % of the passing vehicles.
- *Intervals* define two simulation time steps: *begin* and *end* between which their *Actions* will be performed to reroute entering vehicles.
- *Action* is the abstract supertype of all concrete rerouting *Actions*.
- *DestProb Rerouting* is a rerouting *Action* that assigns every vehicle a new destination *Edge*. If there are multiple *DestProb Rerouting Actions* in one interval, they can be assigned individual *probabilities*. When a new vehicle arrives at the rerouter, one of these *Actions* will be chosen randomly based on the defined *probabilities*.
- *Parking Area Rerouting* is similar to *DestProb Rerouting*, but instead of assigning a new *Edge*, a *Parking Area* is used as new destination.
- *RouteProb Rerouting* is similar to *DestProb Rerouting* but instead of giving vehicles only a new destination *Edge*, a complete new *Route* is assigned.

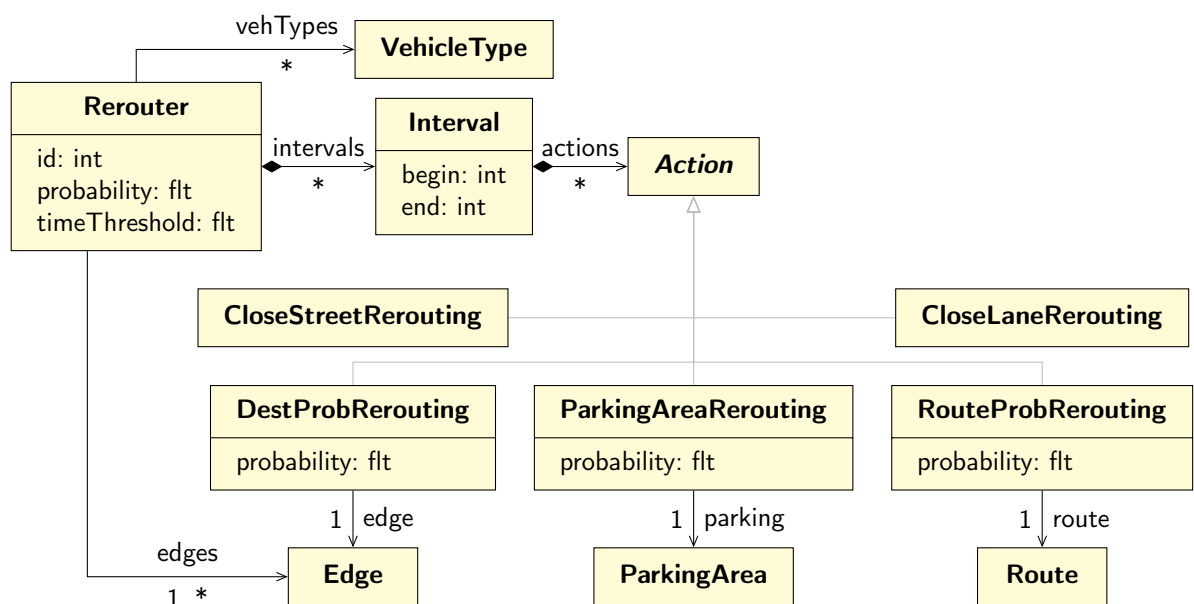
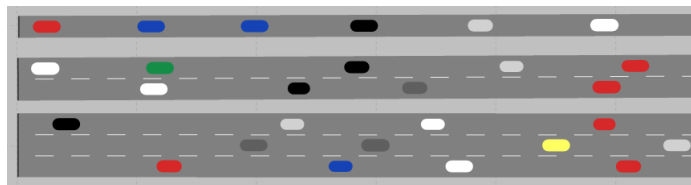


Figure 3.42: SUMO Metamodel: Rerouters

### 3.3.3.2 Traffic Demand in PTV Vissim

In PTV VISSIM, vehicles can be inserted into the road network at the beginning of every link. They are created by so-called 'vehicle inputs' which produce a certain vehicle volume. That volume is time-dependent and can be assigned individual values for different distinct time intervals. Vehicle inputs can insert vehicles of multiple vehicle types. To determine the probability for creating a vehicle of a certain vehicle type, a vehicle composition can be defined for each time interval. Such a vehicle composition defines the relative proportion of the vehicle volume for each vehicle type. Figure 3.43 shows three different vehicle inputs with increasing traffic volume.

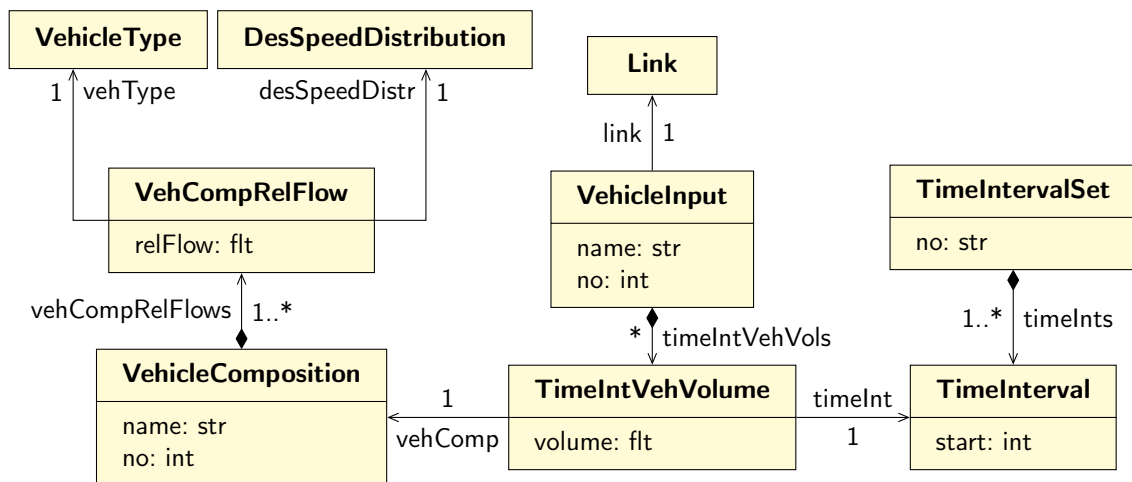


**Figure 3.43:** Three vehicle inputs on three links inserting different traffic volumes - from top to bottom: 1800, 3600 and 5400 vehicles per hour.

Figure 3.44 shows PTV VISSIM's metamodel elements that model vehicle demand. In the following, we will list these elements with some of their properties:

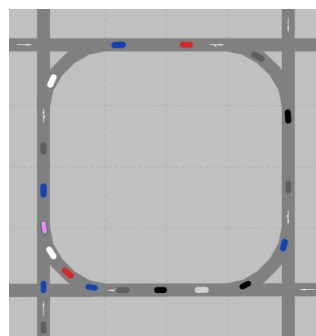
- *Vehicle Inputs* have a *name* and a unique number (*no*) and are placed on a certain *Link*. The input's vehicle volumes are modeled as '*Time Interval Vehicle Volume*' objects, one for each *Time Interval*.
- Each *Time Interval Vehicle Volume* references a *Time Interval* during which the *volume* is inserted into the network. That *volume* is given in vehicles per hour. The referenced *Vehicle Composition* is used to determine which kinds of vehicles are inserted and how likely each kind is to be created.
- In PTV VISSIM, each *Time Interval* belongs to exactly one *Time Interval Set*. All *Time Intervals* of one set must cover the whole time span from the beginning of the simulation to its end. This is why *Time Intervals* only define their *start*. The end of each *Interval* is implicitly defined by the *start* of the next *Interval* in the set. The last *Time Interval* in the set ends at the last simulation second.
- *Time Interval Sets* contain *Time Intervals* that divide the simulation time into disjoint intervals. *Time Intervals* are referenced by multiple metamodel elements to describe the variation of certain values over time. Every usage of these *Time Intervals* can require a different segmentation of the simulation time. This is why there are individual *Time Interval Sets* for different purposes like '*vehicle inputs*' or '*vehicle routes*' that provide a specific segmentation of the simulation time. That purpose is denoted by the unique id (*no*) of each *Time Interval Set*.

- *Vehicle Compositions* have a *name* and a unique number (*no*) and contain a set of *Vehicle Composition Relative Flows* which each define the relative proportion of a *Vehicle Type* to the traffic volume. The sum of their *relative flow* values does not have to equal one. Instead, those *relative flow* values are scaled during the simulation to determine the effective percentage of the traffic volume.
- *Vehicle Composition Relative Flows* define the relative proportion (*RelFlow*) of the referenced *Vehicle Type* to the traffic volume. Furthermore, a *Desired Speed Distribution* is referenced. When vehicles of the referenced *Vehicle Type* are created, they are assigned a desired speed drawn from that distribution.



**Figure 3.44:** PTV VISSIM Metamodel: Traffic Demand

Once vehicles are inserted into the network, they will drive along the road they were created on. As soon as they reach a connector they leave their current link to enter a new one and continue following that new link. Without further guidance, vehicles could for example always take the right turn which results in all vehicles driving in a circle (see figure 3.45). To keep vehicles from always taking the first turn, PTV VISSIM provides multiple routing options that assign explicit routes to vehicles.



**Figure 3.45:** Vehicles without routes take the first connection they find. In this example, all vehicles drive in a circle.

Figure 3.46 shows PTV VISSIM's metamodel elements that model vehicle routing. In the following, we will list these elements with some of their properties:

- *Routing Decisions* have a *name* and a unique number (*no*). They are placed on *Links* at a certain *position*. Vehicles that pass this *position* on the *Link* are affected by the *Routing Decision* and are assigned a new *Route*.
- *Vehicle Routing Decisions* are *Routing Decisions* for private transport vehicles and therefore reference those *Vehicle Classes* that will be affected by the *Routing Decision*.
- A *Route* has a *name* and a unique number (*no*) and consists of a sequence of *Links*. Two consecutive *Links* in that sequence must be connected.
- A *Relative Flow Route* is a *Route* that defines time dependent *relative flow* values. Therefore, *Time Interval Relative Flow* child objects define one *relative flow* value per *Time Interval*. If a *Routing Decision* contains multiple alternative *Routes*, vehicles are assigned one of them and the probability of each alternative depends on the current *relative flow* values of all alternatives.
- *Parking Routing Decisions* assign some vehicles a *Route* to a *Parking Lot*. How many vehicles are sent to a *Parking Lot* is defined by a time dependent *parking rate*. A set of *Time Interval Parking Rates* define those *parking rate* values, one for each *Time interval*. A vehicle's *parking duration* is drawn from a *Time Distribution* which also depends on the *Time Interval*. The *Parking Routing Decision* can contain multiple *Routes* to different *Parking Lots*. However, the way vehicles behave when selecting a parking space will not be considered in this thesis. Figure 3.47 shows an exemplary *Parking Routing Decision* that reroutes a certain percentage of passing vehicles to the two *Parking Lots*.
- *Parking Routes* are *Routes* that lead to a specific *Parking Lot*.
- *Static Routing Decisions* are *Vehicle Routing Decisions* that provide multiple *Static Route* alternatives (see figure 3.48).
- *Static Routes* are *Relative Flow Routes* which means that *relative flow* values are considered when assigning one of the *Static Route* alternatives.
- *Partial Routing Decisions* are local *Routing Decisions* that replace the current *Route* of a vehicle only in a short area. The vehicles will continue their original *Route* once they leave that area. A *Partial Routing Decision* has a fixed *destination Link* and a *destination position* on that *Link*. All its *Partial Routes* must connect the origin *position* (*pos*) and the *destination position* (*destPos*). This can be used to distribute vehicles over short alternative *Routes* that eventually join each other. Figure 3.48 contains an example of such a *Partial Routing Decision* on the left. Approaching vehicles will be distributed over the three alternative paths before they continue their original *Route*.
- A *Partial Route* is a *Relative Flow Route* that connects the the origin *position* (*pos*) and the *destination position* (*destPos*) of a *Partial Routing Decision*.

- A *Partial PT Routing Decision* is a *Routing Decision* for public transport lines (see figure 3.49 for more details). Similar to a *Partial Routing Decision*, it references a *destination Link* and specifies a *destination Position*. With this, public transport vehicles can be lead along different alternative *Routes* within a short area before they continue to pursue their original *Route*.
- *Partial PT Routes* are *Relative Flow Routes* that connect the origin *position (pos)* and the *destination position (destPos)* of a *Partial PT Routing Decision*. For more details see figures 3.49 and 3.51.

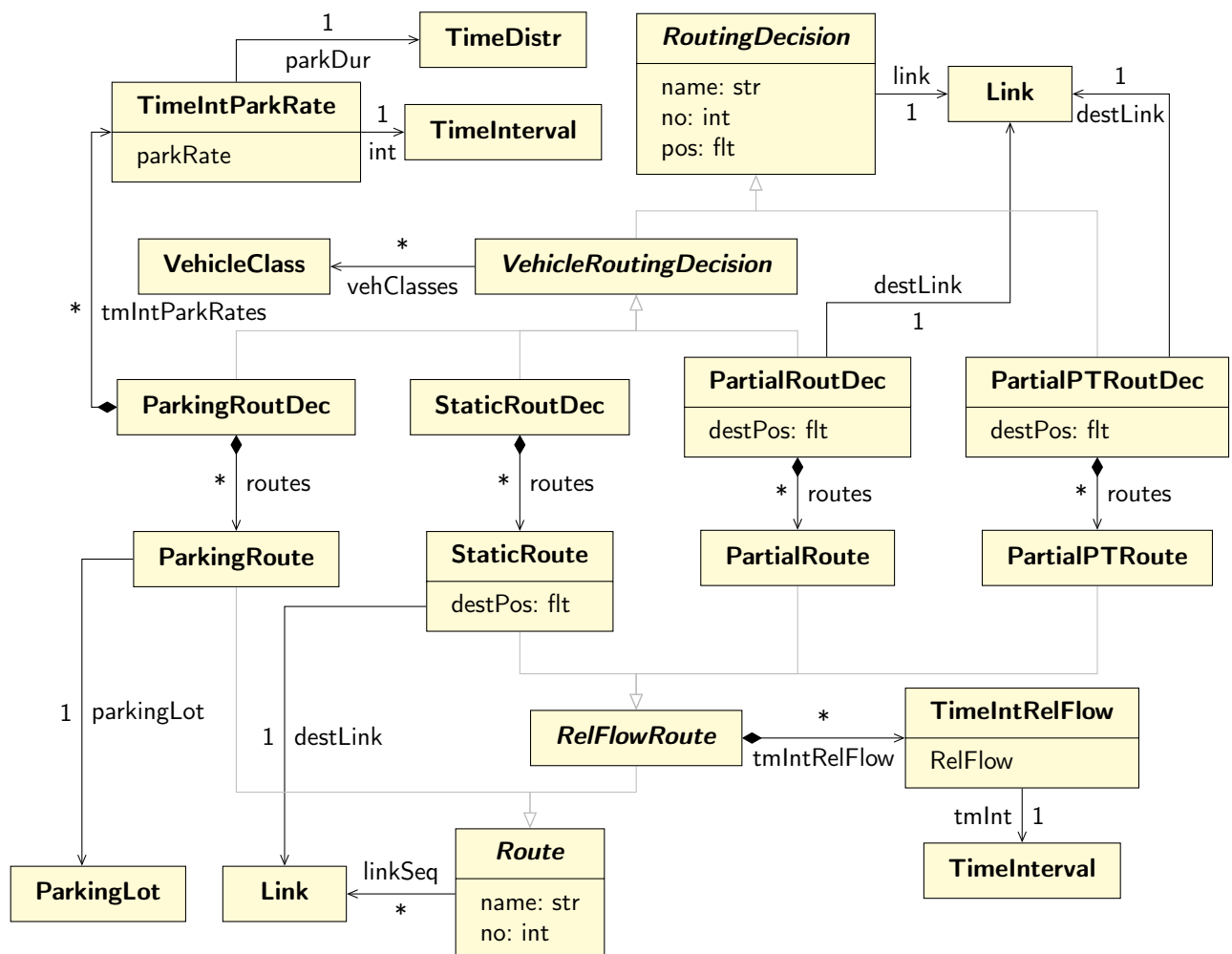


Figure 3.46: PTV VISSIM Metamodel: Vehicle Routes

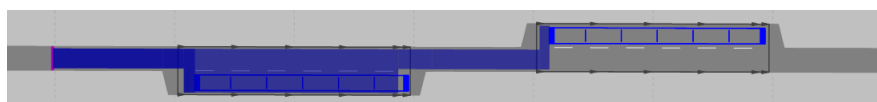
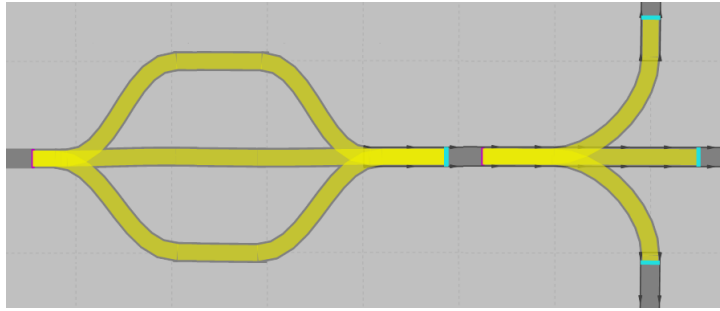


Figure 3.47: A parking routing decision that assigns vehicles a new route to one of the connected parking lots.



**Figure 3.48:** A partial routing decision on the left and a static routing decision on the right.

Public transport is modeled separately from private transport demand in PTV VISSIM. The departure of vehicles in public transport lines can be modeled explicitly. Public transport lines are special routes for vehicles that additionally specify at which public transport stops they halt. Furthermore, PTV VISSIM allows to model the duration the vehicles spend at those public transport stops either by defining a dwell time distribution or by modeling boarding volumes and alighting rates.

Figure 3.49 shows PTV VISSIM's metamodel elements that model public transport. In the following, we will list these elements with some of their properties:

- *PT Lines* have a *name* and a unique number (*no*). They start at the beginning of an *entry Link* and end at an arbitrary *destination position* on a *destination Link*. Like *Routes* (see figure 3.46) *PT Lines* follow a sequence of *Links* that connect the *entry* and *destination Link*. Along that route, the *Line* passes a set of *PT Stops* at which its vehicles can perform *Line Stops* to pick up or drop off passengers. Figure 3.50 shows a *PT Line* that serves three out of four *PT Stops* but leaves out the third one. Every *PT Line* references one *Vehicle Type*. All vehicles created on a *PT Line* belong to that *Vehicle Type* and receive a desired speed drawn from the referenced *Desired Speed Distribution*. *PT Lines* reference a list of *Departure Times* at which vehicles leave the first *PT Stop*. This means that there is a time offset between the vehicle entering the road network at the beginning of the *PT Line* and the vehicle's departure from the first *PT Stop*. Furthermore, the *entry time* is drawn from a *Time Distribution* which is used to vary the time at which public transport vehicles enter the network at the *Line's entry Link*. In this way, the time offset (*entry time offset*) before the first departure should cover the time required to reach the first *PT Stop*, the mean dwell time at that stop and the mean of the *entry time* distribution. By using an offset of zero, the *PT Line's Departure Times* describe the time at which the vehicles will enter the network.
- A *PT Line's Departure Time* describes at which time (*departure*) a public transport vehicle leaves the first stop with respect to the *PT Line's entry time offset*. So a vehicle enters the network at the *Departure Time* minus *entry time offset*. Additionally, a value from the *PT Line's entry time* distribution is added to that entry time.

- *PT Stops* are points where passengers can enter public transport vehicles. To determine how many passengers enter vehicles of a certain *PT Line* at a certain time of day, *PT Stops* contain *Boarding Volumes*. A *Boarding Volume* defines a passenger volume for a specific *PT Line*. Each *Boarding Volume* is valid during a certain time interval [*timeFrom*, *timeTo*]. This can be used to model changing passenger volumes over a long time period.
- A *Halt* represents the action of a public transport vehicle serving a *PT Stop*. It belongs to one *PT Line* and describes that *Line*'s behavior at one specific *PT Stop*. If a *Halt* is *active*, the *Line*'s vehicles stop at the *PT Stop*. The time the vehicles spend at the *Stop* (called dwell time) is either derived from a *Time Distribution* or calculated from the amount of boarding (see *Boarding Volume*) and alighting (*alightPerc*) passengers. The latter option takes properties of the vehicles into account like the boarding time (see figure 3.32).
- *Partial Line Stops* are *Halts* that are not part of a *PT Line* but a *Partial PT Route*. The corresponding *Partial PT Routing Decision* can reference a set of *PT Lines* whose vehicles are affected by the *Routing Decision*. Figure 3.51 contains three *Partial PT Routes*: one performing two *Partial PT Stops*, one skipping the lay-by *Stop* and one skipping both of them.
- *Line Stops* are *Halts* that are part of a *PT Line*.

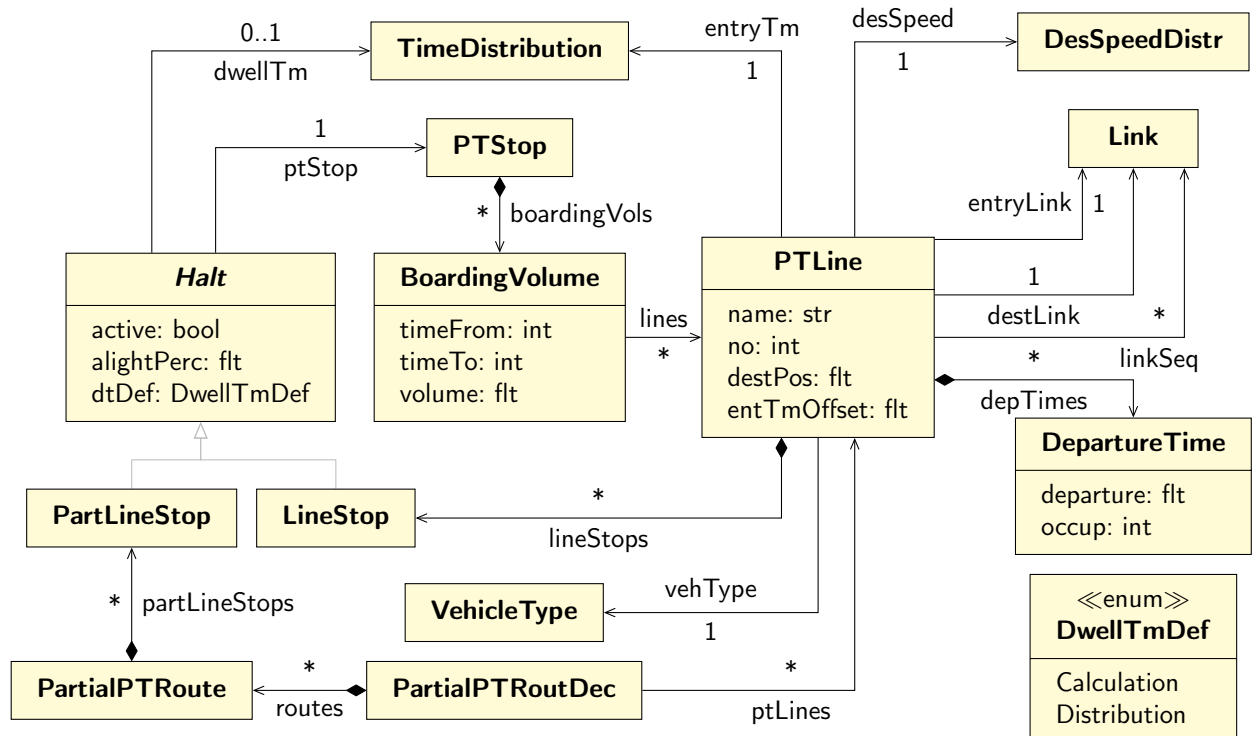
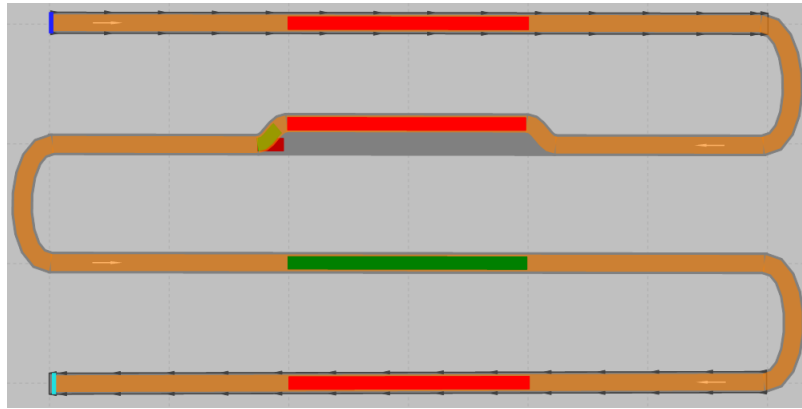
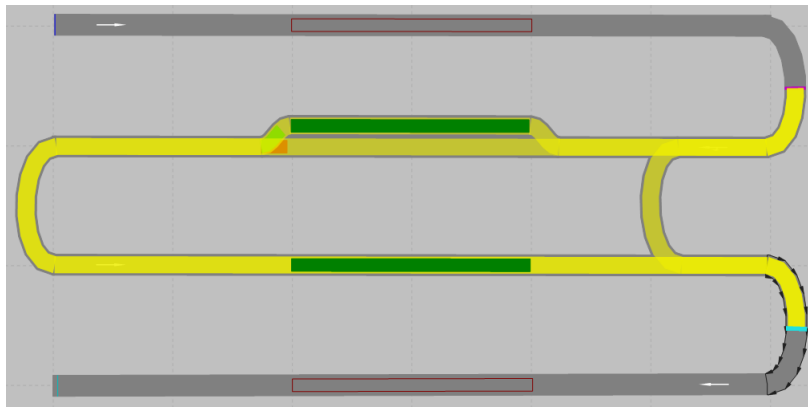


Figure 3.49: PTV VISSIM Metamodel: Public Transport Lines



**Figure 3.50:** A Public transport line serving three out of four stops skipping the third one.



**Figure 3.51:** A partial public transport routing decision that distributes public transport vehicles over three alternative routes to either serve one or two stops or to skip both.



## 3.4 Modeling Output Measurements

In the previous sections, we discussed different aspects of PTV VISSIM's and SUMO's metamodels: transport infrastructure, traffic control and traffic demand. Both programs take instances of their metamodel as input to perform their microscopic traffic simulation. These simulations are usually performed to estimate the quality of traffic facilities. Depending on the investigated infrastructure element, its quality is determined by measuring and calculating certain key figures. According to the HBS<sup>1</sup> [47] (the German guidelines for dimensioning road traffic facilities), the quality of intersections is derived from the vehicles waiting times and a motorway's quality depends on its saturation rate. In the following, we will look at the options that PTV VISSIM and SUMO provide to perform these measurements in the simulation. The output generated by the simulations can then be evaluated and the required characteristics can be computed. The evaluation results can be used to determine the simulated system's quality or to further calibrate the models.

### 3.4.1 Detectors in SUMO

SUMO provides four different types of detectors that can be used as sensors for measurements. The provided detectors are induction loops, instant induction loops, lane area detectors and entry-exit detectors. Induction loops and instant induction loops are placed on a single lane and detect vehicles passing a certain position on that lane. Lane area detectors instead detect vehicles in a larger area and not only at one point on a lane. This detector can be used to detect vehicle queues. Finally, the entry-exit detector is defined by a set of entry and exit points to the detected area. Hence, this Detector can detect vehicles inside an arbitrary area.

Figure 3.52 shows SUMO's metamodel elements that model the different types of detectors. In the following, we will list these elements with some of their properties:

- *Detector* is the abstract supertype of all concrete *Detectors*. Every *Detector* has a unique *id* and a *file* into which the detector writes its results. Additionally, all *Detectors* reference a set of *Vehicle Types* (*vTypes*). Only vehicles of a referenced *Vehicle Type* will be detected by the *Detector*. All other vehicles will be ignored.
- Some of the concrete *Detectors* are placed on a single *lane*. The abstract supertype *Lane Detector* holds the properties which those *Detectors* have in common: Each *Lane Detector* references the *Lane* it is placed on and specifies the *position* where it is located on that *Lane*. Figure 3.53 shows three concrete *Detectors* that are placed on a single *Lane*.

<sup>1</sup>Handbuch für die Bemessung von Straßenverkehrsanlagen

- An *Induction Loop* is a *Lane Detector* that aggregates different figures about passing vehicles. For example it keeps track of the average speed and length of those vehicles as well as the flow that passed the Detector. The *Induction Loop's frequency* determines a time period in which the measured values will be aggregated. These aggregated results are written to the *Detector's file* after each time period.
- Like the *Induction Loop*, the *Instant Induction Loop* is a *Lane Detector*. However, it does not aggregate measured values over a period of time. Instead, it produces output data in every time a vehicle is detected. The *Instant Induction Loop* distinguishes between entering, staying and leaving vehicles. In each simulation step the Detector is occupied, the occupying vehicle's properties are recorded including its id, length, speed and type.
- A *Lane Area Detector* can detect vehicles in an area on one *lane* or on a set of consecutive *lanes*. The start- and end-point of the covered area are given in form of a start position (*pos*) on the first *Lane* and a end position (*endPos*) on the last *Lane*. A *Lane Area Detector* can specify different thresholds: Firstly, a *timeThreshold* is defined, which determines how much time has to pass until a vehicle is considered as halting. Secondly, a *speedTheshold* is given, below which a vehicle's speed has to drop before it is considered as halting. Finally, a *jamThreshold* specifies the maximum distance of a vehicle to its predecessor for the vehicle to be considered as part of the jam. With respect to these thresholds, the *Detector* records various key figures about queues and jams that arise in the observed area. This includes the average speed, time loss and halting duration as well as the mean jam length in that area. Like *Induction Loops*, *Lane Area Detectors* aggregate their measured values for a certain period of time. The Detector's *frequency* defines the length of the measurement intervals. Figure 3.54 shows a *Lane Area Detector* that covers multiple *Lanes*.
- Similar to *Lane Area Detectors*, *Entry Exit Detectors* observe vehicles in a given area. However, instead of restricting the area to a sequence of *Lanes*, *Entry Exit Detectors* define a set of *Entry* and *Exit Points* to the observed area. In this way, *Entry Exit Detectors* are not bound to one lane and therefore they are not *Lane Detectors*. Vehicles passing one of the *Entry Points* (*entries*) will be observed until they leave the area through one of the *Exit Points* (*exits*). Like *Lane Area Detectors*, *Entry Exit Detectors* measure average speed, time loss and the number of halts for passing vehicles in a given time period (*freq*). Additionally, the *Detector* measures the average travel time. To specify when vehicles are considered as halting two thresholds can be defined: *timeThreshold* and *speedThreshold* are used similar to the thresholds of *Lane Area Detectors*. Figure 3.55 shows an exemplary *Entry Exit Detector* with entry points to the west and the south and exit points to the north, south and east.

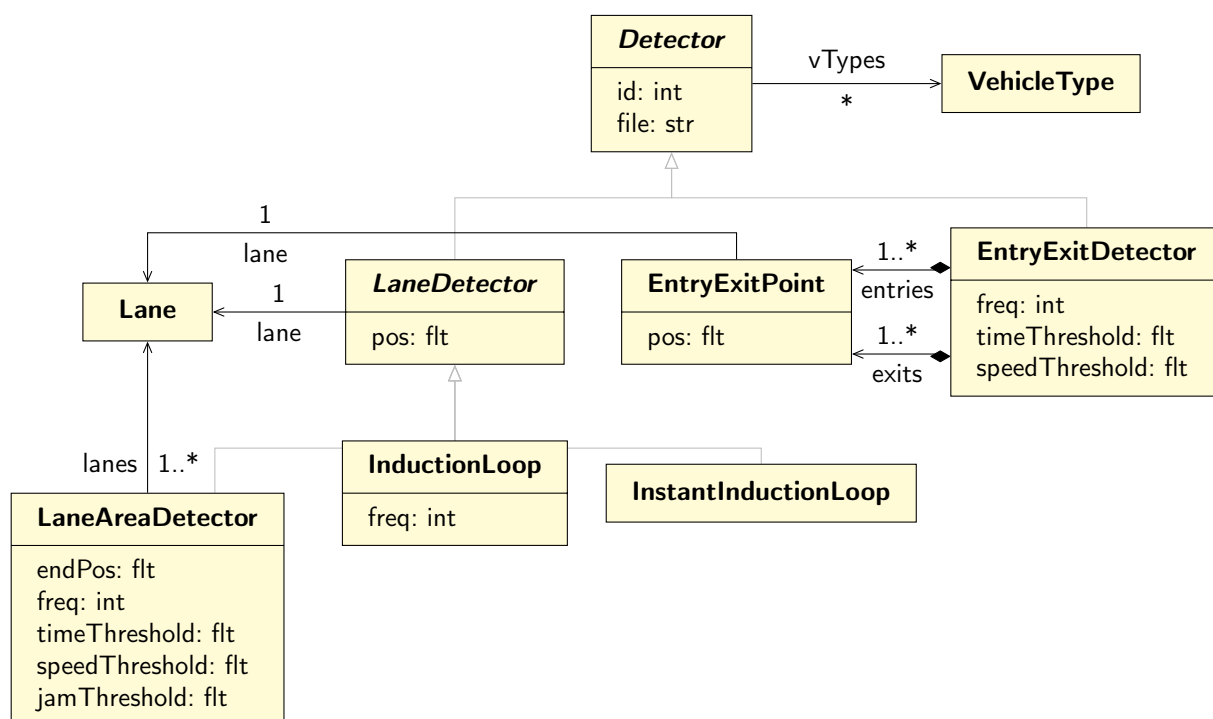


Figure 3.52: SUMO Metamodel: Detectors



Figure 3.53: Three detectors in SUMO that are placed on a single lane. From left to right: instant induction loop (pink), induction loop (yellow) and lane area detector (gray).



Figure 3.54: A lane area detector covering multiple consecutive lanes.

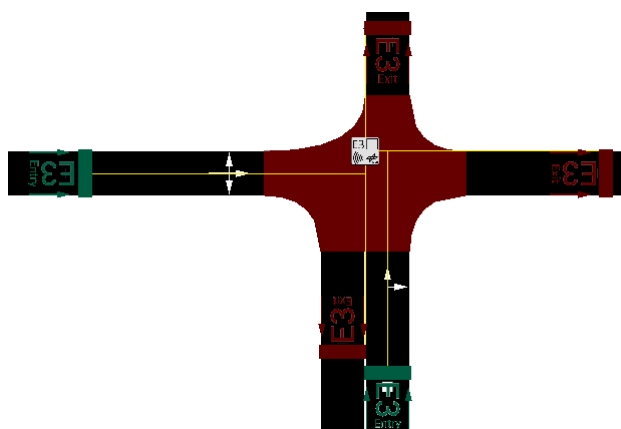
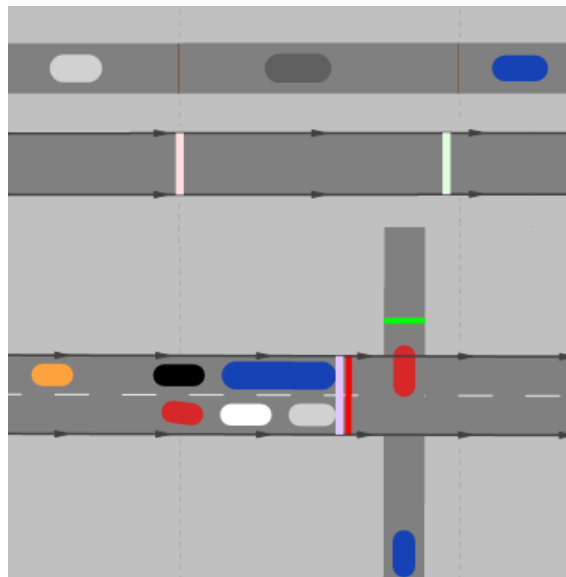


Figure 3.55: An entry-exit-detector detectiong all vehicles that pass an entry point until they leave at an exit point.

### 3.4.2 Measurements in PTV Vissim

PTV VISSIM provides a wide range of data output options to evaluate different aspects of the simulation. For some of them, measurement sensors have to be placed on the road network to record passing vehicles. In the following, we will look at three of those sensors: data collection points which record vehicles passing a certain position on a road, queue counters which record queues of standing vehicles and travel time measurements which record vehicles in a certain area and compute their travel times.

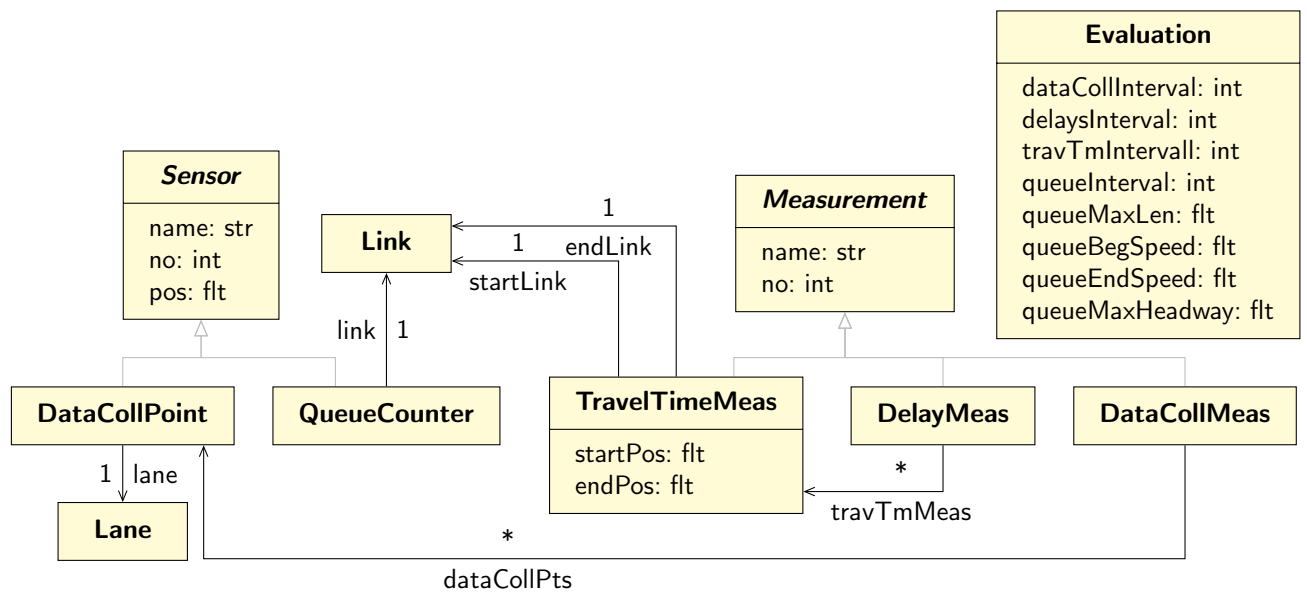


**Figure 3.56:** Three different measurement options in PTV VISSIM - from top to bottom: data collection points (green lines), a travel time measurement (pink and light green lines) and a queue counter (light purple line).

Figure 3.57 shows PTV VISSIM's metamodel elements that model measurements. In the following, we will list these elements with some of their properties:

- *Sensors* have a *name* and a unique number (*no*) and are placed onto the road network at a certain *position* on a *Link* or *Lane*. The behavior of a *Sensor* and the kind of data it records depends on the concrete type.
- *Data Collection Points* are *Sensors* that are placed on *Lanes*. They detect properties of all passing vehicles like their acceleration, length or passenger count. The top *Link* in figure 3.56 contains two exemplary *Data Collection Points*.
- *Queue Counters* are *Sensors* that are placed on *Links*. They detect the length of standing vehicle queues as well as the number of vehicles in those queues. The crossing in figure 3.56 contains an exemplary *Queue Counter* that detects the queue in front of the signal head.

- *Measurements* have a *name* and a unique number (*no*). They record detected data to produce the simulations result files. These files can then be used to evaluate the simulation. If a *Measurement* is performed in multiple simulation runs, the minimum, maximum and average values as well as the standard deviation of the collected data are computed.
- *Travel Time Measurements* cover an area from a *start position* on a *start Link* to an *end position* on an *end Link*. They record the time vehicles need to travel through that area as well as the distance they traveled in it. Figure 3.56 contains such a *Travel Time Measurement* on the *Link* in the middle. Vehicles are detected as soon as they enter the area on the left side. Their travel time is computed when they leave it on the right side.
- *Delay Measurements* combine multiple *Travel Time Measurements*. Every *Travel Time Measurement* can be part of multiple *Delay Measurements*. *Delay Measurements* record the passing vehicle's average number of stops (without halts at public transport stops or parking lots) as well as their average standstill time. Furthermore, the delay of vehicles in the *Travel Time Measurement* areas is computed as the difference between their actual travel time and the ideal travel time in those areas.
- *Data Collection Measurements* combine multiple *Data Collection Points* to collect their data in one *Measurement*. *Data Collection Points* can be part of multiple *Data Collection Measurements* at the same time. The recorded data is aggregated over all passing vehicles per time interval.
- Every PTV VISSIM model has exactly one *Evaluation* object that models some global properties for all sensors and measurements in a network. This includes the time intervals in which the measured data is aggregated. Furthermore, properties of *Queue Counters* can be specified globally. The *maximum queue length* (*queueMaxLen*) that can be detected by *Queue Counters* can be specified. Also three thresholds can be defined that determine when vehicles are considered as part of the queue. The *maximum headway* (*queueMaxHeadway*) is the maximum distance of a vehicle to the one in front of it. Also the speed below which vehicles are considered as part of the queues tail (*queueBegSpeed*) can be defined. A vehicle is considered as part of a queue, until its speed rises above the *end speed* (*queueEndSpeed*).



**Figure 3.57:** PTV VISSIM Metamodel: Measurements

## 4 Model Comparison

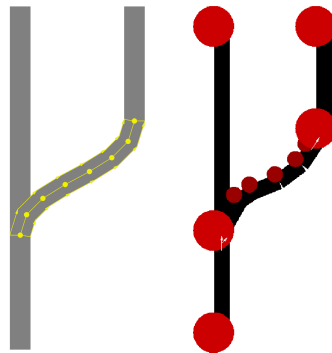
In chapter 3 we developed two metamodels describing the most important elements of PTV VISSIM and SUMO models. In this chapter, we will compare the two metamodels and describe how similar concepts relate to each other. Since PTV VISSIM and SUMO model some concepts in very different ways, we will focus on these major differences. The following sections provide possible solutions on how to overcome these major differences. For some of the problems, we will give multiple solutions with different benefits and drawbacks. A prototype implementation of the relations given in this chapter is described in chapter 5.

### 4.1 Comparing Roads and Intersection

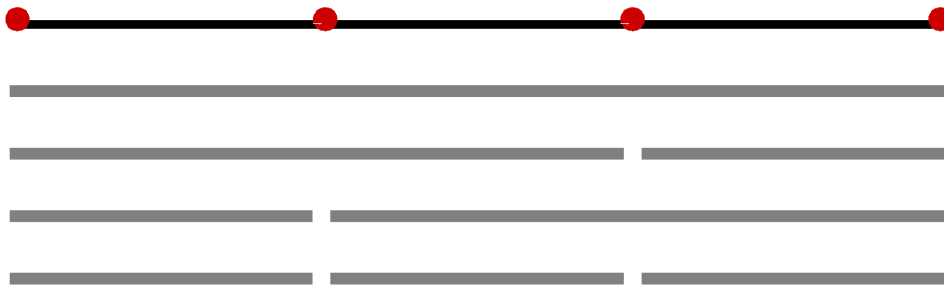
The road network is the foundation of both PTV VISSIM's and SUMO's models on which the remaining model elements are built upon. Even in this critical part, there are major differences between PTV VISSIM and SUMO models. While SUMO models the road network as a directed graph with nodes (junctions) and edges, PTV VISSIM does not possess a concept similar to nodes.

In SUMO, vehicles can move from one edge to another at junctions where incoming and outgoing lanes are connected via connections. In PTV VISSIM, however, vehicles can leave a link via connectors that can start at arbitrary positions on a link. Connectors can also end at any position on the destination link. This flexibility of PTV VISSIM allows to create a branch-off in the middle of the link without having to divide the main link into two links as shown in figure 4.1. In SUMO, the edge has to be split into two before a third edge can branch in the middle.

Considering the example in figure 4.1, one link in PTV VISSIM relates to one or more edges in SUMO and connectors also relate to edges. In the example above, the link representing the main road corresponds to two edges in SUMO. This exemplary PTV VISSIM network can be extended, such that - for any given number  $N > 0$  - one link corresponds to  $N$  edges in a derived SUMO network. Furthermore, when considering the opposite direction of the relation, a SUMO network merely consisting of a sequence of  $N$  edges can result in  $2^{N-1}$  possible corresponding PTV VISSIM networks. Figure 4.2 shows an exemplary edge sequence in SUMO consisting of three edges and all possible representations in PTV VISSIM: from one link representing all three edges to one individual link per edge.



**Figure 4.1:** A road with a branch-off modeled in PTV VISSIM (left) and in SUMO (right). PTV VISSIM uses one link to model the main road and one connector to model the branch-off while SUMO requires three edges.

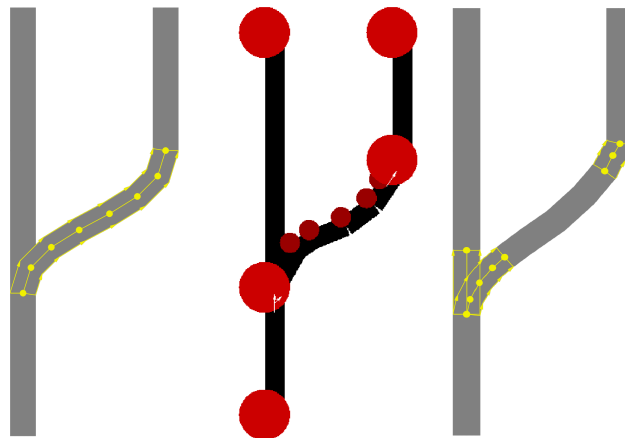


**Figure 4.2:** A sequence of edges in SUMO (top) and the four possible representations in PTV VISSIM below where one link in PTV VISSIM can represent multiple consecutive edges in SUMO.

Aside from the nondeterministic transformation of edge sequences to links, the proposed relation of connectors and edges poses another problem. Consider again the exemplary SUMO network in the middle of figure 4.3. This SUMO network is derived from the PTV VISSIM network on the left as explained above. By simply transforming edges to links, we receive the PTV VISSIM network shown on the right. The resulting PTV VISSIM model on the right differs from the original PTV VISSIM model on the left. Especially the branch, which used to be a connector (marked in yellow), is a link after the second transformation. This violates the `PUTGET` property of a bidirectional transformation (see section 2.1.4). Since we try to find such a bidirectional transformation, another relation is required for connectors, links and edges.

An alternative correspondence in SUMO for connections in PTV VISSIM are connections which would solve the `PUTGET` violation. Connectors in PTV VISSIM and connections in SUMO have similar properties, however, connectors are more flexible since they can branch and merge links at arbitrary positions. Connections in contrast can only connect lanes at the end of an edge. Therefore, connectors cannot simply be transformed into connections without further restrictions of PTV VISSIM. The transformation of connections in SUMO to connectors in PTV VISSIM is possible without restrictions.

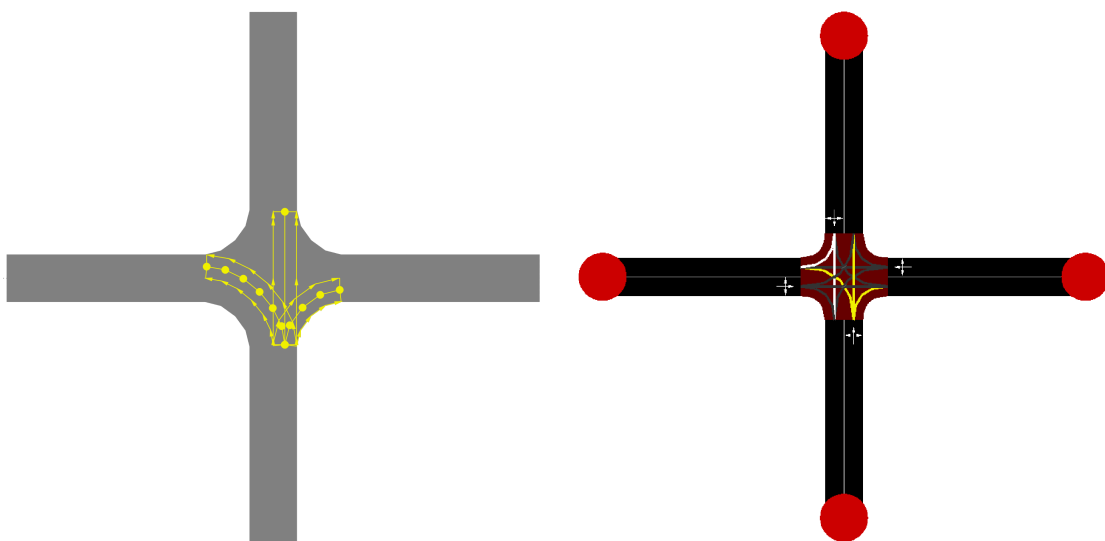




**Figure 4.3:** A road with a branch-off modeled in PTV VISSIM on the left and the derived SUMO in the middle. The PTV VISSIM model on the right is derived from the SUMO model by transforming edges to links.

The flexibility of PTV VISSIM's links and connectors can be restricted to avoid the exponential number of possible representations of edge sequences as well as the described PUTGET violation. Every connector in PTV VISSIM may only branch and merge links at their end points. So, wherever there is a branching road, the SUMO model contains a junction while in the PTV VISSIM model all links must end before resp. start behind the intersection so the branching links can be connected via connectors.

This restriction implies a one-to-one relationship between links in PTV VISSIM and edges in SUMO. Furthermore, connectors in PTV VISSIM solely correspond to connections in SUMO, which also only exist between the end of one and the beginning of another edge (see figure 4.4).



**Figure 4.4:** An intersection in PTV VISSIM (left) with three exemplary selected connectors (yellow) and the corresponding connections in SUMO (left).

The restriction of valid PTV VISSIM models can increase the number of links and connectors required to model a given road section. However, it does not restrict the set of possible geometries that can be modeled. Since connectors have the same properties as links in PTV VISSIM (see section 3.1.1.2), every road section that a user would model as a connector can be replaced by link with the same polygon points and two short connectors.

An alternative way to cope with those nondeterministic representations would be to apply correspondence heuristics like partitioning the network into edge sequences without branches of maximum length. However, most of the remaining model elements have lane relative positions. Therefore, we will use the more simple one-to-one relation. So for the following relations between PTV VISSIM and SUMO we will assume the described restriction of PTV VISSIM models.

With these relations between infrastructure elements in PTV VISSIM and SUMO, we can now look at some minor differences.

First of all, links in PTV VISSIM have a gradient attribute which can be set independent of the modeled geometry. SUMO does not contain a corresponding attribute. On the other hand, edges can be assigned a length that does not equal the length of the modeled position vector. In PTV VISSIM, a link's length is always derived from its geometry and cannot be set to a custom length. To achieve a similar behavior on corresponding roads, these attributes should be restricted from changing. However, being able to change gradients in PTV VISSIM or lengths in SUMO can be seen as unique features of both tools which motivate switching from one to the other in the first place. In this second scenario, both attributes can simply be ignored.

Junctions in SUMO have no correspondence in PTV VISSIM, however, their position corresponds to the start- and endpoints of one or more links. When connecting two - previously separate - links in PTV VISSIM via a connector, the junctions in SUMO corresponding to end of the first and the start of the second link have to be joined to one junction. When joining junctions, it is important to keep all connections that already exist. Also, the attributes of the new junction must be derived from the two original junctions. Furthermore, all references to one of the original junctions need to be updated.

Lanes in PTV VISSIM and SUMO are very similar. They are contained in links resp. edges which have a one-to-one relation as described above. Furthermore, lanes have an index and a width in both PTV VISSIM and SUMO. This means that a lane  $l_s$  in SUMO corresponding to a lane  $l_v$  in PTV VISSIM are consistent iff the edge  $e$  containing  $l_s$  and the link  $l$  containing  $l_v$  correspond to each other. Furthermore  $l_s$  and  $l_v$  must have the same index and width.

As established above, connectors in PTV VISSIM correspond to connections in SUMO. Connectors in PTV VISSIM can connect multiple adjacent lanes of one link with the same amount of adjacent lanes on another link. However, connections in SUMO can connect

lanes only pairwise. Thus, each connector corresponds to one or more connections. Also, each connection corresponds to exactly one lane of a connector.

In PTV VISSIM, a link's polygon points describe the shape of the link's middle. In SUMO, an edge's shape describes its left side. Furthermore, lanes in PTV VISSIM do not explicitly model their shape. Instead it is derived from the link's shape. In SUMO, lanes and edges both model their shape explicitly, however, the shapes of all lanes are recalculated by the SUMO network editor when saving the network.

The polygon points of a link can be computed from an edge's position vector and vice versa by shifting the points by half the sum of all lane widths towards the middle resp. the left side. It suffices to keep the link's geometry consistent with the corresponding edge's shape since the SUMO editor will always recalculate the shape of all lanes. When creating a new lane in PTV VISSIM, a corresponding lane can be created in SUMO with a temporary approximation derived from the shape of the edge. This approximation will be overwritten once the SUMO network is saved in the editor.

Levels in PTV VISSIM have no exact correspondence in SUMO, but they influence the z-coordinate of positions. If a link at level  $z_v$  in PTV VISSIM has a polygon point with a z-offset  $z_o$ , the corresponding position in SUMO has the z-coordinate  $z_s = z_v + z_o$ .

The relations presented in this section form the foundation on which the relations in the following sections build upon. Especially relations of network objects that are placed onto links or edges require the relations of this section.

## 4.2 Comparing Speed limits

In this section, we will discuss how speed limits in SUMO relate to desired speed decisions in PTV VISSIM.

In SUMO, speed limits are explicitly modeled as attributes of lanes and connections. In PTV VISSIM however, speed limits are neither modeled explicitly nor modeled as attributes of links, lanes or connectors. Instead, vehicles in PTV VISSIM receive their desired speed when they are inserted into the network, which they keep for the rest of their trip unless they drive past a desired speed decision.

This means that wherever two consecutive edges have different speed values in SUMO, the corresponding PTV VISSIM model has a desired speed decision which assigns vehicles their new desired speed. Creating a new desired speed decision in PTV VISSIM does not only affect the speed on the link where it is created but also on all reachable links up to the next desired speed decision. This means a desired speed decision relates to the speed values of a subgraph of SUMO's network. Furthermore, it is possible to create two desired speed decisions on two separate links which eventually merge into one link. In this scenario,

vehicles with desired speeds from both decision points can drive on the same merged link. This behavior is not possible in SUMO since every lane has exactly one speed value.

To resolve this conflict, either each speed value in SUMO is derived (e.g. as mean) from multiple desired speed decisions in PTV VISSIM or the set of valid PTV VISSIM models has to be restricted. The former option would allow PTV VISSIM users to use its full flexibility and to model differences in desired speeds more detailed. However, once a speed value in SUMO is derived from multiple values in PTV VISSIM, it is impossible to propagate future changes to that speed value back to PTV VISSIM. The latter option could be realized by automatically creating a new desired speed decision, wherever the reachability trees of two desired speed decisions meet each other.

Another problem arises due to the semantic difference between PTV VISSIM's desired speeds and SUMO's driver independent speed limits. In SUMO, the reactions of vehicles to the speed limit of an edge are modeled as speed factors. A speed factor is an attribute of a vehicle type. It is multiplied with the speed limit of an edge to compute the desired speed of vehicles on that edge. If a vehicle has a speed factor of 1.05, it will always try to drive 5 % faster than the speed limit. The speed factors in SUMO are drawn from a normal distribution similar to desired speeds in PTV VISSIM. These desired speed values are also drawn from a distribution and model the vehicle's reaction to a speed limit. Also, each desired speed decision can define individual speed distributions for each vehicle class.

As a first simple relation, the mean value of a desired speed distribution in PTV VISSIM for a vehicle class can be set to the speed limit of the corresponding edge times the speed factor of the corresponding vehicle type.

However, the distribution from which the speed factors are drawn is an attribute of a vehicle type in SUMO and vehicles cannot change their vehicle type during their trip. In PTV VISSIM the desired speed distribution can change at every desired speed decision. So the mean of the speed factor distribution can correspond to the mean values of multiple speed distributions. The same applies to the standard deviation of the speed factor distribution.

Deriving the speed factor distribution from multiple - possibly different - desired speed distributions makes it impossible to propagate their changes back to PTV VISSIM. This problem can either be resolved by prohibiting changes to the speed factor distribution in SUMO or by allowing PTV VISSIM and SUMO to model vehicle speeds individually.

Aside from desired speed decisions, PTV VISSIM also provides reduced speed areas in which vehicles reduce their desired speed. Here again the problem occurs, that SUMO models speed limits per edge. If a reduced speed area is situated in the middle of a link, there is no way to model the same speed reduction in SUMO without altering the geometry of the network. As a solution, the set of valid PTV VISSIM models can be restricted. Alternatively, PTV VISSIM models can be allowed to model vehicle speeds with more detail than SUMO.

In the former solution, models are considered valid if all reduced speed areas cover the entire link they are placed on. In the latter solution, reduced speed areas in PTV VISSIM can be used to compute an edge's speed limit. For example, the speed inside and outside of the reduced speed area can be summed up and weighed proportionally to the length of the link they apply on. However, this is only an approximation and can produce different simulation results in PTV VISSIM and SUMO.

Finally, SUMO allows to model variable speed signs which apply on a set of lanes. Their speed changes at specific times which are defined as vss steps. In PTV VISSIM variable speed signs can be modeled with desired speed decisions which can define a time interval during which they apply. For each vss step and each lane of a variable speed sign in SUMO, there is one desired speed decision in PTV VISSIM.

This relation raises a problem since desired speed decisions already relate to speed values of lanes. Transforming variable speed signs to desired speed decision which themselves are transformed back to speed values of lanes would violate the PUTGET rule (see section 2.1.4). Hence, there must be a distinction between desired speed decisions that correspond to edge speed limits and those that correspond to variable speed signs. This required distinction can be found in the time interval during which desired speed decisions apply. Those that are active during the complete simulation correspond to speed limits of edges while desired speed decisions that only apply for a shorter period of time correspond to vss steps of variable speed signs in SUMO.

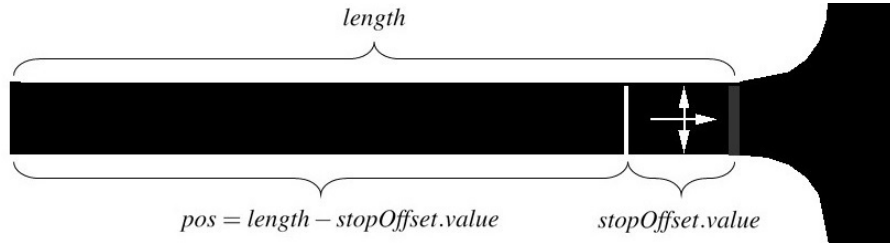
## 4.3 Comparing Stop Lines

Apart from speed signs, PTV VISSIM and SUMO can also model other traffic signs that indicate the right of way at intersections like stop signs, priority signs or signal heads. These signs are always accompanied by a stop line. In this section, we will discuss how the positions of stop lines in PTV VISSIM relate to stop lines in SUMO. The traffic control aspects of the corresponding signs will be discussed in sections 4.5 and 4.6.

In PTV VISSIM, stop signs, priority signs and signal heads can be placed at any position on any lane. The position, where these objects are placed is also the position where vehicles will wait until they may drive. Thus, the position of the sign is also the position of the stop line.

SUMO does not model traffic signs and signal heads explicitly. Instead, they are implicitly placed at the entrance to a junction. A junction's type determines which kind of signs are placed on the incoming lanes. Nevertheless, the position of a stop line can be modeled explicitly in the form of a stop offset. Such a stop offset specifies the distance of a stop line from a junction.

Therefore the positions of traffic signs in PTV VISSIM relates to stop offsets in SUMO. More specifically, a stop sign's position in PTV VISSIM equals the length of the link it is placed on minus the value of the corresponding stop offset as shown in figure 4.5. If no stop offset is specified in SUMO, the distance between a stop line and the junction is zero.



**Figure 4.5:** Consistency of Stop Lines in PTV VISSIM and SUMO.

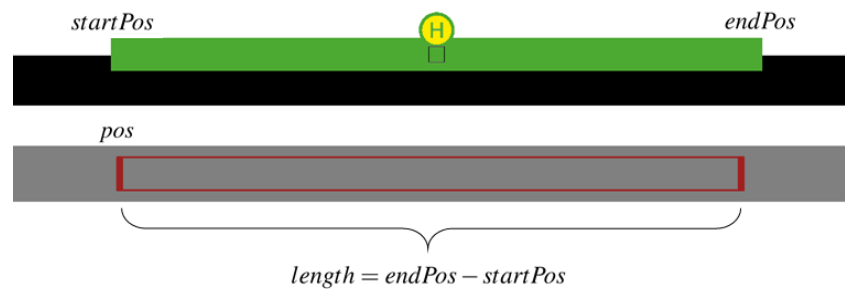
In both PTV VISSIM and SUMO, stop lines can be restricted to a set of vehicle classes that must respect the sign and wait at the stop line. A stop line applies to all vehicle classes in SUMO that correspond to vehicle classes of the corresponding traffic sign or signal head in PTV VISSIM. In SUMO, vehicle classes that are not contained in a stop offset of a lane, implicitly use the stop line at the entrance to the junction. In PTV VISSIM, however, the corresponding vehicle classes do require an explicitly modeled sign or signal head. Otherwise, vehicles of those classes will cross the intersection without paying respect to any sign.

#### 4.4 Comparing Stop Infrastructure

PTV VISSIM and SUMO both support parking lots for private transport vehicles and stops for public transport vehicles. In SUMO, both facilities are concrete specialization of the abstract type *StopInfrastructure*. Similarly, public transport stops and parking lots in PTV VISSIM are subtypes of *ObjectOnLanes*. Both abstract supertypes define the lane on which the facility is placed as well as the position on that lane at which it starts.

Parking lots and stops both cover a certain area of the lane they are placed on. PTV VISSIM models that area with the start position on the lane and a length attribute. In SUMO instead, the covered area is given by the start- and end-position of the parking lot or stop. Thus, its length is equal to the difference of its end- and start-position.

The consistency of public transport stops in PTV VISSIM and bus stops in SUMO in terms of their position in the network can be defined as follows: A bus stop  $bs_s$  in SUMO and a corresponding public transport stop  $pts_v$  in PTV VISSIM are consistent if the lane  $bs_s.lane$  in SUMO corresponds to the lane  $pts_v.lane$  in PTV VISSIM. Furthermore the position  $bs_s.startPos$  must be equal to the position  $pts_v.pos$  and  $pts_v.length$  must equal  $bs_s.endPos - bs_s.startPos$  as shown in figure 4.6. The consistency of parking lot positions in PTV VISSIM and SUMO is defined analogously.



**Figure 4.6:** Consistency of PTV VISSIM public transport stops and SUMO bus stops.

In SUMO, bus stops can contain a set of accesses, which allow public transport vehicles to access the bus stop from lanes other than its own. PTV VISSIM does not have a corresponding concept.

Parking lots can be placed onto the road in both PTV VISSIM and SUMO. The length of the parking spaces inside the parking lot can be specified in both tools. SUMO additionally allows modeling the width of the parking spaces as well as the angle between them and the lane which is not possible in PTV VISSIM. PTV VISSIM in turn allows modeling opening hours and a maximum parking duration which cannot be modeled in SUMO. To keep parking lots in PTV VISSIM and SUMO consistent, all of these attributes must receive fixed default values. The width of parking lots in SUMO must equal the lane's width and its angle must be zero. In PTV VISSIM, a parking lot has to be open during the whole simulation time and vehicles must be allowed to park at a parking lot for the entire simulation.

If these unique features motivate switching from one tool to the other, these attributes can simply be ignored when keeping parking lots consistent.

Additionally to parking lots on the road, SUMO supports parking areas that are situated next to the road. Inside these areas, individual parking spaces can be modeled explicitly. In PTV VISSIM, there is no concept of parking areas next to the road. PTV VISSIM does provide abstract parking lots which remove vehicles from the road network. However, these abstract parking lots are origin and destination points for simulating dynamic assignment. They cannot be used as destinations for parking routes while in SUMO parking areas can serve as such a destination. Hence, abstract parking lots in PTV VISSIM are no suitable correspondence to parking areas in SUMO.

Parking areas could be modeled in PTV VISSIM by creating a link at the position of each individual parking space inside a parking area in SUMO. These links can then be connected to the main road and ordinary 'on-road' parking lots can be placed onto them. However, SUMO does not model the paths on which the vehicles drive to their parking spaces. Hence, it would require complex heuristics for generating the geometry of connectors between the link and the parking lot. We will not pursue this approach in this thesis. Instead we will focus on keeping 'on-road' parking lots consistent.

## 4.5 Comparing Right of Way at Intersections

Traffic control is modeled very differently in PTV VISSIM and SUMO. While in PTV VISSIM, traffic control signs can be placed at any position in the network, SUMO defines the right of way per junction. Thus, a junction's type in SUMO can relate to multiple signs or signal heads in PTV VISSIM. In this section, we will discuss 5 important junction types without signal control in SUMO and how they relate to right of way rules in PTV VISSIM.

PTV VISSIM provides stop signs, priority signs (priority rules) and conflict areas to model the right of way at intersections. Furthermore, signal heads can be used to model signal-controlled intersections (see section 4.6). In SUMO, a junction's type as well as its priority type define the right of way. The SUMO network editor guesses which connections have the right of way and how conflicting connections are treated based on those two types. This initial right of way definition can be altered by changing the state attribute of connections or the response sets of requests. For each connection at a junction it, contains exactly one request. Each request references one of the connections and describes its conflicts with other connections (foes) and how they are resolved. The response-set of a request contains those conflicting connections that get the right of way over the request's connection.

A conflict of two connections in SUMO corresponds to a conflict area in PTV VISSIM. Every conflict area references two conflicting links or connectors and defines on which of them vehicles have to wait. This conflict area status relates to an entry in the response set of the connection corresponding to the yielding link.

As an alternative to conflict areas, PTV VISSIM provides priority rules to model the priorities of conflicting streams. However, the PTV VISSIM manual advises to always "model the standard priority rules for conflicting traffic flows which are not controlled by signals by means of conflict areas" [1, p. 541] and to use priority rules only "if conflict areas do not produce the desired results and if you have sufficient experience in modeling with priority rules" [1, p. 541]. Hence, we will keep conflict areas consistent with SUMO's requests and leave priority rules as a unique feature of PTV VISSIM.

Let conflict area  $C_v$  in PTV VISSIM correspond to two conflicting connections  $C_s^1$  and  $C_s^2$  with requests  $R_s^1$  and  $R_s^2$  in SUMO and let  $R_s^1.connection$  correspond to  $C_v.link1$  and  $C_s^2$  correspond to  $C_v.link2$ . Then  $C_v$  is consistent with the conflict of  $C_s^1$  and  $C_s^2$  if either  $R_s^1.response$  contains  $C_s^2$  and  $C_v.status = OneYieldsTwo$  or  $R_s^2.response$  contains  $C_s^1$  and  $C_v.status = TwoYieldsOne$ . If neither  $R_s^1.response$  contains  $C_s^2$  nor  $R_s^2.response$  contains  $C_s^1$ , the conflict is unresolved in which case  $C_v = Passive$ .

Since conflict areas are computed from overlapping links, PTV VISSIM models can contain conflict areas that do not correspond to conflicts in SUMO. For example, wherever multiple connectors leave the same lane to connect it with lanes in different directions, these connectors



will overlap and therefore create conflict areas in PTV VISSIM. SUMO, however, does not consider the corresponding connections as conflicts. As a solution, all conflict areas between connectors origination from the same link will not be kept consistent with SUMO. Instead, these conflict areas receive the status *Undetermined* which ensures that vehicles on these connectors will still take note of other vehicles until they leave the conflict area. The status *Undetermined* is different from the status *Passive* since the latter status makes vehicles ignore vehicles on the other connector.

Aside from resolving conflicts, conflict areas also specify other properties that influence the behavior of vehicles at the intersection. In SUMO, these properties can be specified globally for an entire connection instead of each conflict individually. Thus, PTV VISSIM allows for a more detailed model of the intersection than SUMO, which is why the set of attribute values in PTV VISSIM have to be aggregated into single values for SUMO. Alternatively, the set of valid PTV VISSIM models could be restricted to those models that have the same attribute value for each conflict area of a connector.

For example, each conflict area in PTV VISSIM defines a visibility distance for both links which define the distance from the intersection at which vehicles will be able to see oncoming vehicles on the conflicting link. A connection in SUMO has a single visibility value which applies to all conflicts. From a safety point of view, the minimum visibility distance in PTV VISSIM should be used in SUMO to evaluate if the intersection is safe even with the shortest visibility distance. However, if the goal is to evaluate the capacity of the intersection it would make more sense to use a weighted mean in SUMO. Each visibility distance in PTV VISSIM can be weighed with the vehicle volume on the link. This reduces the error produced by vehicles that have a shorter or larger visibility distance in SUMO than in PTV VISSIM.

Another example is the specification of whether vehicles try to avoid blocking the intersection. Again, this can be modeled per conflict area in PTV VISSIM and per connection resp. junction in SUMO.

The Boolean attribute *cont* of a request states whether vehicles on the request's connection will drive into the junction even if they cannot cross it completely. Additionally, the *keepClear* attribute of a connection states whether vehicles will avoid blocking the junction.

In PTV VISSIM, the attributes *avoidBlockMinor* and *avoidBlockMajor* of a conflict area can be used to model that behavior on the two conflicting links. The Boolean *avoidBlockMajor* values can be aggregated for minor connections in SUMO. If one of these values is *true*, the connection's *keepClear* value is *true* as well and the *cont* value of the connection's request is *false*. In this way, if vehicles avoid blocking one conflict area on a connector in PTV VISSIM, they will not enter the junction in SUMO unless they can cross it without blocking it.

For major connections in SUMO, the *avoidBlockMinor* values of the corresponding conflict areas in PTV VISSIM can be aggregated. These values are probabilities, which state the probability that a vehicle on the major link will avoid to block the intersection. They have to be aggregated into a single Boolean value. For example, the *keepClear* value of a major connection in SUMO can be set to *true* if one of the conflict areas surpasses a certain threshold (e.g. 50 %). The *cont* value of the connections request is set to *false* accordingly.

The relation of conflict areas in PTV VISSIM to foes and response sets in SUMO can be used for any junction type. In the following, we will discuss how specific junction types relate to PTV VISSIM.

- Right before left On junctions of the type *right before left*, all connections have the state *equal*. The response sets of the junction's requests are selected in a way such that vehicles will always yield to vehicles coming from their right side. These conflicts in SUMO relate to conflict areas in PTV VISSIM as described above.
- Allway stop On junctions of the type *allway stop*, every incoming lane has an implicit stop sign and all vehicles must halt at the stop line before crossing the junction. All connections receive the status *wait*. The conflicts within the junction relate to PTV VISSIM's conflict areas as described above. To ensure that vehicles in PTV VISSIM will also halt before crossing the intersection, stop signs have to be placed on every incoming lane. The position of these signs relates to SUMO's stop offsets as described in section 4.3.
- Priority On junctions of the type *priority*, there are major and minor connections. Connections with the state *minor* must yield while connections with the state *major* have the right of way. The response sets of a priority junction's request are selected in a way such that major connections get the right of way. The conflicts within the junction relate to PTV VISSIM's conflict areas as described above.
- Priority Stop Junctions of the type *priority stop* behave like priority junctions with the addition that vehicles on minor connections must halt before crossing the junction. Those minor connections receive the status *stop*. The corresponding lanes in PTV VISSIM must contain stop signs to ensure that vehicles will also halt in PTV VISSIM. The position of these signs relates to SUMO's stop offsets as described in section 4.3.
- Zipper On junctions of the type *zipper*, multiple connection from different incoming lanes on the same edge merge into one outgoing lane. These merging connections have the state *zipper*. The junction's conflicts and the response sets of its requests relate to PTV VISSIM as described above.

---

## 4.6 Comparing Traffic Light Programs

In section 4.5 we discussed how the right of way rules at non-signal-controlled intersections in PTV VISSIM and SUMO relate to each other. In this section, we will look at signal-controlled intersections. Similar to stop signs and priority signs in the previous section, signal heads are placed implicitly at the entrance to a junction in SUMO (see section 4.3). In PTV VISSIM, signal heads are modeled explicitly.

It is possible, that a lane that ends in a signal-controlled intersection does not have a signal head in PTV VISSIM. In that case, the connections in SUMO that originate from this incoming lane are uncontrolled. This relation does not apply in the other direction in general. If for example one of three connections that originate from the same lane in SUMO is uncontrolled, the corresponding signal head in PTV VISSIM cannot simply be removed. This is because the stop line is situated on the incoming lane and not on the connectors themselves which correspond to the connections in SUMO. Thus, all connectors that originate from the same lane in PTV VISSIM adhere to one signal head while in SUMO each connection has its own signal. To be able to keep the signal controls of PTV VISSIM and SUMO consistent, the set of valid SUMO models must be restricted to those models where all connections with the same origin lane always receive the same signal. Furthermore, either all of them are uncontrolled or none of them are. This restriction does not limit any important abilities of SUMO. On the contrary, this restriction ensures that only reasonable signal programs are valid. Without the restriction, it is possible to create a signal program in SUMO in which traffic turning left receives the green signal and traffic turning right on the same lane receives the red signal. However, these situations do not occur in reality. For safety and comprehensibility reasons every lane always receives one signal at a time.

Signal-controlled junctions and their connections reference the signal program by which they are controlled (unless they are uncontrolled). For the following relations between signal programs in PTV VISSIM and SUMO, it is important that all connections of a junction reference the same signal program. Accordingly, all signal heads at an intersection in PTV VISSIM may only reference signal groups of the same signal controller.

The smallest logical unit in a PTV VISSIM signal program is a signal group which controls multiple vehicle streams. SUMO instead does not contain the concept of signal groups. In SUMO, the smallest unit is a connection.

In PTV VISSIM, signal heads display the current signal of a signal group. As described above, all connections in SUMO that originate from the same lane must always receive the same signal. Hence, these connections correspond to the same signal group in PTV VISSIM. With this relation between signal groups in PTV VISSIM and connections in SUMO, we can now develop the relations between signal programs in PTV VISSIM and SUMO.

A signal program in PTV VISSIM describes a signal changing over time for each signal group. Such a signal program is either based on signal groups or stages. In SUMO, a signal program consists of multiple phases (which do not correspond to stages in PTV VISSIM). Each phase describes the displayed signals of all signal groups for a certain duration. During such a phase, no signal can change.

This means that whenever a connection is to receive a new signal, the signal program has to define the start of a new phase. Writing all the phases of a signal program one below the other produces a matrix. Each row of that matrix describes the signals of one connection. Table 4.1 shows an example of such a signal program matrix. Each row represents one phase of a signal program in SUMO and specifies its duration. One connection may receive the same signal in two consecutive phases in case one of the other connections receives a new signal. Hence the duration of a signal group's signal in PTV VISSIM equals the summed durations of consecutive phases during which the signal of the corresponding connection does not change. The signal group corresponding to connection 5 in table 4.1 receives the red signal for 45 seconds since the connection's signal is red during the first two phases in SUMO.

connection	1	2	3	4	5	6	7	8	9	10	11	12
duration 42 s	G	G	g	r	r	r	G	G	g	r	r	r
3 s	y	y	y	r	r	r	y	y	y	r	r	r
42 s	r	r	r	G	G	g	r	r	r	G	G	g
3 s	r	r	r	y	y	y	r	r	r	y	y	y

**Table 4.1:** An exemplary signal program matrix in SUMO. The signal program controls a junction with 12 connections. Each column in the matrix contains the signals of one connection. SUMO abbreviates signals as follows: G - green major, g - green minor, y - amber, r - red

PTV VISSIM distinguishes between free signals and signals with a fixed duration. SUMO does not know this distinction. In general, the signals amber and red-amber have a fixed duration. Therefore, we will assume that amber and red-amber signals in SUMO relate to fixed signals in PTV VISSIM while all other signals relate to free signals.

The signal sequence which each signal group in PTV VISSIM must reference equals the run-length encoding of the corresponding connection's column in the phase-matrix. For the exemplary connection 5 in table 4.1 the run-length encoding results in  $[(r, 45), (G, 42), (y, 3)]$ .

A signal program's cycle time is not modeled explicitly in SUMO. Instead, it is derived from the duration of all its phases. The exemplary signal program in table 4.1 has a cycle time of 90 seconds. In PTV VISSIM however, the cycle time is modeled explicitly. PTV VISSIM's signal programs contain free signals which have a start point but no duration. To ensure

that a program's last signal is not displayed for the rest of the simulation, a cycle time has to be specified. Once the cycle time is reached, the signal program will start over. The start of a signal program can be shifted by an offset in both PTV VISSIM and SUMO since it is a cyclic program. This offset allows to synchronize signal programs of consecutive intersections.

In PTV VISSIM each signal program also contains an intergreen matrix. The matrix does not influence the displayed signals, however, it is used to validate the signal program. An intergreen time is the time required for one stream to leave the intersection before the next stream may enter it. If the time between the green signals of two conflicting signal groups is too short, their intergreen time could be violated. SUMO does not model intergreen times. Hence, if a signal program is changed in SUMO, it could violate the intergreen times modeled in PTV VISSIM, even though it is considered valid in SUMO. To avoid this situation, the intergreen matrix of a signal program in PTV VISSIM can also be considered when changing the corresponding signal program in SUMO. Thus, changes in a signal program in SUMO that would violate an intergreen matrix in PTV VISSIM can be detected and denied.

Next to signal group based signal programs, PTV VISSIM also supports stage-based signal programs. One stage combines multiple signal groups that receive the green signal at the same time while all other signal groups receive the red signal. A stage-based program only models the times at which the stage changes. The actual signals during the stage transition are modeled in so-called interstage programs which can be modeled in SUMO similar to the signal group based signal programs as described above. The duration of the SUMO phase corresponding to the time between the stage transitions is equal to the time difference between their starts minus the length of the first interstage program.

PTV VISSIM and SUMO both allow switching the signal program of an intersection during the day or the week. A WAUT<sup>1</sup> in SUMO relates to a daily program in PTV VISSIM. Each program start in PTV VISSIM relates to a WAUT switch in SUMO. Both reference corresponding signal programs and equal starting times. A daily program in PTV VISSIM may schedule the first signal program at any time. Hence, it is possible that at the start of the simulation no signal program is running at an intersection. This is not possible in SUMO since every WAUT must reference a start program. To achieve similar behavior in SUMO, an additional signal program is required in which all signals are off.

Even though an intersection is controlled by a signal program, it is still possible that two conflicting streams can drive at the same time. For example, if two opposite streams receive the green signal at the same time, those vehicles that are turning left still have to yield to the opposite stream. These conflicts have to be resolved explicitly in both PTV VISSIM and SUMO. To do so, SUMO distinguishes between different types of green signals: *green major* and *green minor*. Streams that do not have to worry about others get the *green major* signal.

---

<sup>1</sup>Wochenschaltautomatik (~ weekly switch automatism)

Streams with the *green minor* signal pay respect to other streams and yield if necessary. The conflicts of connections with the signal *green minor* are resolved with response sets as described for conflicts of non-signal-controlled intersections in section 4.5. So these conflicts can be modeled with conflict areas in PTV VISSIM.

Sometimes, vehicles turning right receive an individual lane and are not controlled by a signal head. Instead, they respect a green-arrow sign. In PTV VISSIM, such a sign is modeled as a stop sign that references a signal group. The stop sign is active while the referenced signal group has the red signal which makes vehicles halt and observe vehicles of other streams before crossing the intersection. In SUMO, the corresponding connection receives the signal *stop*. If the signal group has the green signal, vehicles turning right do not have to worry about other streams. This is why the stop sign in PTV VISSIM is inactive during that time. In SUMO the corresponding connections receive the green signal.

### 4.7 Comparing Vehicle Properties

Vehicle properties are modeled as attributes of vehicle types in both PTV VISSIM and SUMO. Some of these properties have a direct correspondence in both models. For example, the boarding time of a public transport vehicle type in PTV VISSIM equals the boarding duration in SUMO or the person capacity of a vehicle type in SUMO equals the capacity in PTV VISSIM.

However, other properties that both models share are modeled very differently in PTV VISSIM and SUMO. The most noticeable difference is that PTV VISSIM often models vehicle properties in the form of a distribution. Each time a vehicle of a certain vehicle type is created, its properties are drawn from the distributions of that vehicle type. In SUMO, most of these properties are modeled as single values.

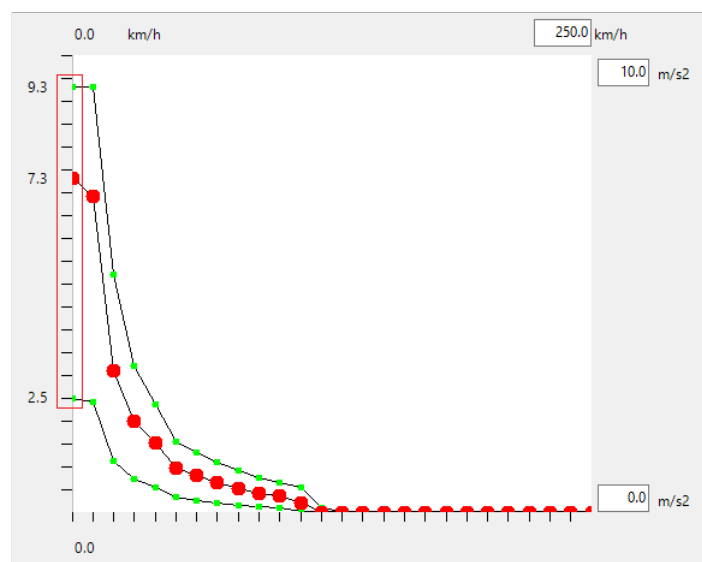
For example, vehicle types in SUMO have a color attribute. Vehicle types in PTV VISSIM do not have a single color but a color distribution instead. One possible solution to keep these two concepts consistent is aggregating the distribution in PTV VISSIM to a single value in SUMO. For example the color with the highest probability in a distribution can be used in SUMO. Alternatively, the color distribution in PTV VISSIM can be modeled in SUMO by using SUMO's vehicle type distribution. Such a vehicle type distribution combines a set of vehicle types, each of which have a certain probability. Also, a vehicle type distribution can be used in SUMO as if it was a concrete vehicle type. Each of the concrete vehicle types can be assigned one of the colors from the distribution in PTV VISSIM as well as its probability. In this way, a vehicle type in PTV VISSIM corresponds to a vehicle type distribution in SUMO which contains multiple concrete vehicle types.

Properties like the length and width of a vehicle type can be modeled explicitly in SUMO.

In PTV VISSIM, these values are derived from the 3D models of vehicles. Furthermore, not all vehicles of the same vehicle type have the same length and width since a vehicle's 3D model is drawn from a distribution. The PTV VISSIM metamodel developed in chapter 3 does not consider visualization aspects of PTV VISSIM which is why it does not contain the concepts of 3D models and their distributions. Hence, the width and length properties can be seen as an individual modeling feature of SUMO. Alternatively, modifications can be forbidden for these attributes. In the former case, the length and width properties can simply be ignored by the consistency rules. In the latter case, default values depending on the vehicle class can be used for width and length in SUMO. These default values can be chosen as the average width and length of the corresponding vehicle class in PTV VISSIM. For example, passenger cars have a length of 4.3 meters and a width of 2 meters. In this way, default values can be defined for every vehicle type in SUMO.

Besides color and shape, PTV VISSIM has further distribution based vehicle properties. One of them is the maximum acceleration function, which describes a vehicle's maximum acceleration depending on its speed. In SUMO, each vehicle type has only a single acceleration value.

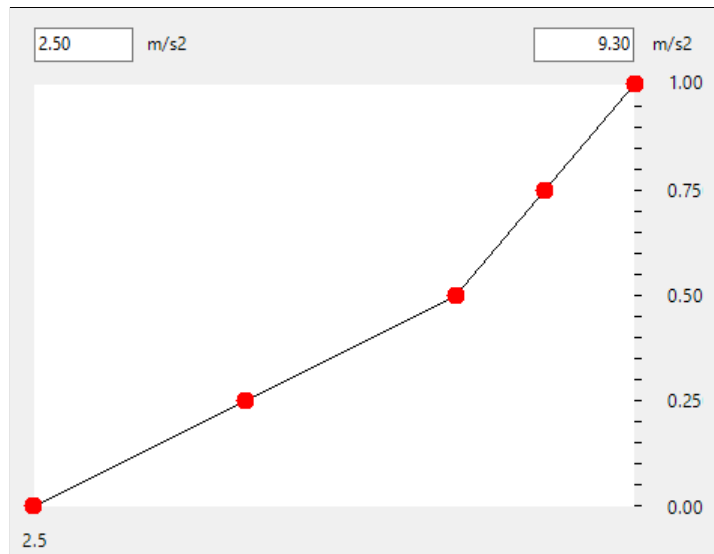
The decrease in maximum acceleration with increasing speed cannot be modeled explicitly in SUMO's static data model. However, the decrease is considered in SUMO's dynamic driving behavior models. For example, the Krauss model [39] assumes a linear decrease of the maximum acceleration. The acceleration value given in a (static) SUMO model is the vehicles' acceleration when standing still. Thus, a vehicle type's acceleration value relates to the value of PTV VISSIM's acceleration function at  $0 \frac{km}{h}$ , as shown in figure 4.7.



**Figure 4.7:** A maximum acceleration function in PTV VISSIM: maximum acceleration over velocity. Merely the highlighted data points at  $0 \frac{km}{h}$  correspond to the acceleration attribute of vehicle type in SUMO.

This reduces the relation from an entire function down to one function value. However, PTV VISSIM's acceleration function does not model a single function but a distribution of functions (see section 3.3.1.2). So the maximum acceleration of a vehicle type at  $0 \frac{km}{h}$  is not a single value but rather consists of a median value as well as a lower and upper bound. When a vehicle is created, its maximum acceleration is the result of interpolation between the median and upper or lower bound depending on a uniformly distributed random variable.

This distribution can either be aggregated to a single acceleration value in SUMO or the same approach that was used for the color distribution can be used again. In the former case, the median value of an acceleration function at  $0 \frac{km}{h}$  can be used. In the latter case,  $n$  values can be selected for which the interpolation will be calculated. This produces  $n$  acceleration values for  $n$  concrete vehicle types that are part of a vehicle type distribution. Since the random variable in PTV VISSIM is uniformly distributed, each of the  $n$  random values has the same probability. Hence, each of the interpolation results has the same probability which is why all concrete vehicle types have a probability of  $\frac{1}{n}$ . The maximum deceleration of a vehicle type in PTV VISSIM relates to the emergency deceleration of a vehicle type in SUMO accordingly.



**Figure 4.8:** A distribution of maximum acceleration values at  $0 \frac{km}{h}$

Figure 4.8 shows the distribution of the acceleration function from figure 4.7 at  $0 \frac{km}{h}$ . To approximate this distribution in SUMO, five vehicle types can be created with different acceleration values. This forms a discretization of the distribution. Selecting the equidistant probabilities  $[0, 0.25, 0.5, 0.75, 1]$  results in the acceleration values  $[2.5, 4.9, 7.3, 8.3, 9.3]$  as shown in Figure 4.8.

Since vehicle types in PTV VISSIM contain multiple distribution based attributes, each of them can be modeled in SUMO using multiple concrete vehicle types and a vehicle type distribution. Discretizing  $m$  distributions into  $n$  values each results in  $n^m$  possible



combinations of attribute values, hence it results in  $n^m$  concrete vehicle types in SUMO. The probability of each concrete vehicle type in SUMO can be computed as the product of its attribute's probabilities in their respective distributions in PTV VISSIM. Table 4.2 shows the probabilities of all possible combinations of an exemplary discretization of three distributions. In the example, two deceleration values, four acceleration values and three colors were used.

decel		$-7 \frac{m}{s^2}$			$-8 \frac{m}{s^2}$		
	color	red	green	blue	red	green	blue
accel	$2.5 \frac{m}{s^2}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$
	$4.9 \frac{m}{s^2}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$
	$7.3 \frac{m}{s^2}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$
	$8.3 \frac{m}{s^2}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$
	$9.3 \frac{m}{s^2}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{30}$

**Table 4.2:** 30 possible parameter value combinations and their probabilities for three parameters: *accel*, *decel* and *color*. Each *accel* value has a probability of  $\frac{1}{4}$ , each *decel* value has a probability of  $\frac{1}{2}$ . The colors have the following probabilities: red -  $\frac{1}{2}$ , green -  $\frac{1}{6}$  and blue -  $\frac{1}{3}$ .

Aside from the vehicle type's acceleration and emergency deceleration, SUMO also models the maximum speed of vehicles. This property is not modeled explicitly in PTV VISSIM but it relates to the speed  $v_{max}$  at which the maximum acceleration function becomes  $0 \frac{m}{s^2}$ . When a vehicle reaches  $v_{max}$ , its maximum acceleration is  $0 \frac{m}{s^2}$  hence it can never reach a higher speed than  $v_{max}$ .

Vehicle types can be combined in vehicle classes in both PTV VISSIM and SUMO. In both tools, vehicle classes can be used to restrict lanes to a certain kind of vehicles. To do so, the allowed vehicle classes can be referenced instead of every single vehicle type that belongs to one of them. In SUMO each vehicle type belongs to exactly one vehicle class the set of vehicle classes is fixed and cannot be extended by the user. In PTV VISSIM vehicle types can belong to multiple vehicle classes and it is possible to create new vehicle classes. To keep vehicle classes in PTV VISSIM and SUMO consistent, the set of valid PTV VISSIM models has to be restricted to those where each vehicle type belongs to exactly one vehicle class. Also, a valid PTV VISSIM model may only contain those vehicle classes that have a correspondence in SUMO. Additionally to vehicle classes, PTV VISSIM's vehicle types belong to a vehicle category which defines its basic behavior. SUMO does not model vehicle categories. However, PTV VISSIM's vehicle categories can be derived from the vehicle class of a vehicle type. For example, vehicle types of the classes passenger, taxi or hov belong to the category car and bicycles, mopeds and motorcycles belong to the category bike.

## 4.8 Comparing Driving Behavior

Additionally to vehicle properties, PTV VISSIM's and SUMO's vehicle types also model some driving behavior properties. In PTV VISSIM the desired acceleration and deceleration are attributes of vehicle types. Similar to the maximum acceleration, these two properties are modeled as acceleration and deceleration functions. The desired deceleration of a vehicle type in PTV VISSIM relates to the deceleration property of a vehicle type in SUMO. They can be kept consistent similar to the maximum acceleration as described in section 4.7. The desired acceleration in PTV VISSIM has no correspondence in SUMO.

Next to desired acceleration and deceleration, vehicles in PTV VISSIM have a desired speed. However, it is not modeled as an attribute of vehicle types since it depends on the link a vehicle drives on. The relation between SUMO's speed limits and PTV VISSIM's desired speeds is discussed in detail in section 4.2.

A lot of driving behavior properties in PTV VISSIM are link dependent. They are either attributes of driving behavior objects which assigned to links or they are modeled directly as attributes of network objects. In SUMO, these properties are modeled as attributes of vehicle types which is why they cannot change during a vehicle's trip. This makes it impossible to recreate the PTV VISSIM behavior in SUMO.

The set of valid PTV VISSIM models could be restricted to those where the same driving behavior is used on all links. However, this would considerably limit PTV VISSIM's modeling power. On the other hand, aggregating all used driving behavior values in PTV VISSIM to single attribute values of vehicle types in SUMO would make it impossible to propagate changes to those attributes back into PTV VISSIM. Prohibiting changes to driving behavior attributes in SUMO would considerably limit SUMO's modeling power.

Moreover, these driving behavior values are often carefully chosen to calibrate the models so that the simulation recreates measured characteristics. Therefore, the PTV VISSIM and SUMO models should be calibrated individually, which is why we will not develop consistency rules for these attributes.

## 4.9 Comparing Traffic Demand

PTV VISSIM and SUMO both allow to insert vehicles at various points into the network during the simulation. These entry points are modeled as vehicle inputs in PTV VISSIM. They are situated at the beginning of a link and can produce individual vehicle volumes during different time intervals. In SUMO, flows are used to create vehicles during the simulation. Those vehicles either follow a specific route or travel from an origin edge to a destination edge.

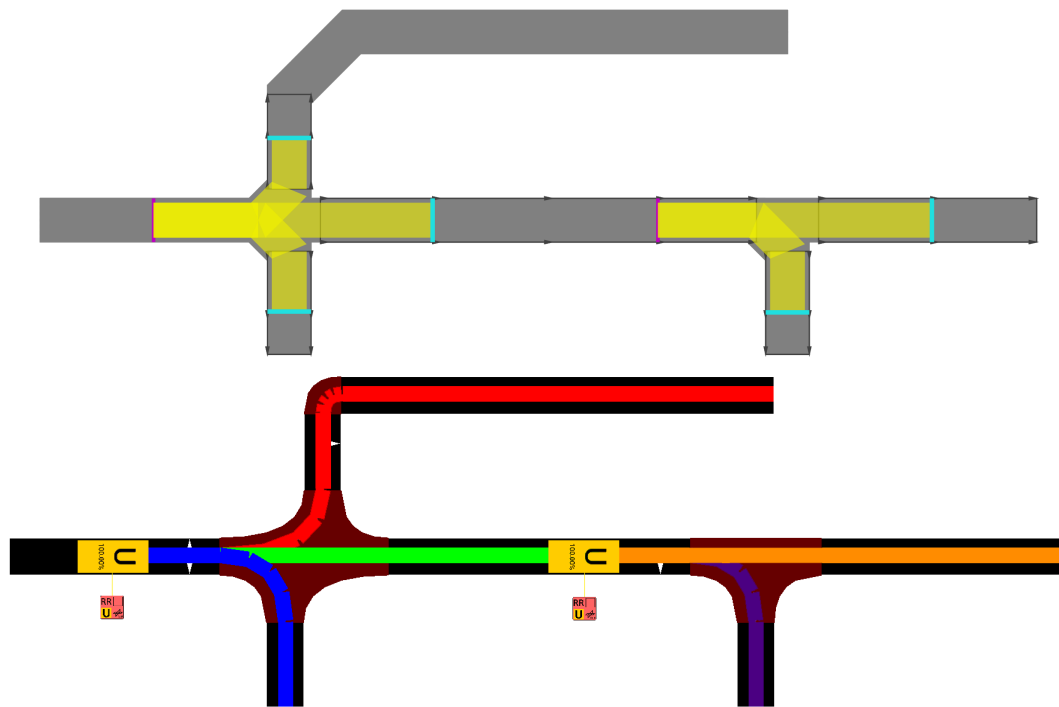
SUMO supports four different kinds of flows: *number* flows, *period* flows and *vehiclesPerHour* flows insert vehicles at equidistant points in time into the network. The fourth kind is *probability* flows, which specify the probability for a vehicle to be created in one time step. This flow type inserts vehicles with random time gaps between them. In PTV VISSIM vehicles are always inserted with random time gaps. Hence, there is no correspondence to the first three flow types. The probability of the fourth flow type relates to the expected amount of vehicles per hour of a vehicle input in PTV VISSIM.

The major difference between PTV VISSIM's and SUMO's demand metamodel is that in SUMO, all vehicles receive a route to their destination when they are created. As soon as a vehicle in SUMO reaches its destination, it is removed from the network. In PTV VISSIM vehicles usually do not have a destination. Instead, they always follow the next connector they encounter and leave the simulation only when they reach the end of a link that is not connected to any other link.

It is possible to assign routes to vehicles in PTV VISSIM using vehicle routing decisions. These routing decisions are typically used in the immediate vicinity of intersections. Routing decisions are placed on all incoming lanes and define routes for each possible turning movement at the intersection. Furthermore, they specify how many arriving vehicles will choose each route. Those routes typically end just behind the intersection. Vehicles that have crossed the intersection will start again to follow the next possible connector until they reach the next routing decision. This is a typical way to recreate turning probabilities in PTV VISSIM that have been measured in the real world.

These local routing decisions can be modeled with rerouters in SUMO. Figure 4.9 shows two routing decisions in a PTV VISSIM network and the corresponding SUMO rerouters below. Unlike in PTV VISSIM, the possible route-options of a rerouter should not end just behind the junction since vehicles will be removed from the simulation once they have reached the end of their newly assigned route. This is because the rerouter overwrites the routes of passing vehicles instead of merely replacing it temporarily as PTV VISSIM does.

By traversing the graph formed by SUMO's junctions and edges, the rest of the routes can be computed. Starting from the edge that corresponds to the last link of a route-option in PTV VISSIM, the graph can be traversed until a dead-end junction or another rerouter is reached. For example, the red route in figure 4.9 continues after it has passed the junction until it reaches the dead-end. The green route instead ends at the first edge behind the junction since that edge already contains the next rerouter. When encountering a junction with multiple turning movements during the traversal, a heuristic for selecting one outgoing lane can be applied. For example, to achieve the same route that vehicles would take in PTV VISSIM, the connection with the highest id number can be selected.



**Figure 4.9:** Two routing decisions (blue lines) with multiple route options (highlighted in yellow) in a PTV VISSIM network and the corresponding SUMO routes and rerouters below.

Both SUMO and PTV VISSIM allow to restrict rerouting or routing decisions to a certain time interval and a specific set of vehicle types resp. vehicle classes. A time interval in PTV VISSIM only defines its beginning. The end of a time interval is always one time step before next time interval in the time interval set. So a time interval in PTV VISSIM either corresponds to the beginning or the end of a time interval in SUMO. In PTV VISSIM routing decisions of one type (e.g. static) use intervals from the same interval set which contains distinct intervals. In SUMO, however, time intervals of different rerouters can overlap. Hence, one - possibly overlapping - time interval in SUMO corresponds to one or more distinct time intervals in PTV VISSIM. The correspondence between parking area rerouters in SUMO and parking routing decisions in PTV VISSIM are defined analogously.

When defining the relation between rerouters in SUMO and routing decisions in PTV VISSIM as described above, the question remains how the initial routes of flows in SUMO can be modeled in PTV VISSIM. In general, PTV VISSIM's routing decisions would be a suitable correspondence. However, routing decisions already correspond to rerouters in SUMO which would result in two possible correspondences in SUMO when creating a routing decision in PTV VISSIM. Therefore, the relation would be non-deterministic and would violate the PUTGET property of a bidirectional transformation.

In a typical use case, a traffic engineer uses data from a traffic census which contains the absolute number of vehicles per turning movement at an intersection. With those absolute

probabilities, the routing decisions can be modeled at each intersection. It is rarely the case that traffic engineers know the entire route of a vehicle when working with measured data. Assuming that the routing decisions will be explicitly modeled for all intersections in a network, the initial routes of flows in SUMO can be ignored and do not have to be modeled in PTV VISSIM.

Additionally to vehicle flows, SUMO also allows modeling the movement of single vehicles. This is not possible in PTV VISSIM, at least not for private transport vehicles. However, SUMO's vehicle movements and trips are suitable to model public transport vehicles that correspond to public transport lines in PTV VISSIM. More precisely, a public transport line in PTV VISSIM corresponds to the route of a vehicle movement or trip in SUMO. A public transport line in PTV VISSIM can be served by several vehicles. Each departure of such a public transport line in PTV VISSIM corresponds to one vehicle movement or trip in SUMO with a corresponding departure time. So one public transport line in PTV VISSIM corresponds to multiple vehicle movements or trips in SUMO.

PTV VISSIM allows to model partial routes for vehicles of public transport lines. Like the routing decisions for private transport vehicles, these partial public transport routes correspond to rerouters that only apply to public transport vehicle classes.

Every public transport line and partial public transport route in PTV VISSIM contains line stops at public transport stops along the route. These relate to stops in the corresponding route in SUMO. Line stops in PTV VISSIM and stops in SUMO both reference a public transport or bus stop. In PTV VISSIM, the duration of the stop can either be calculated from given boarding and alighting rates or drawn from a distribution. In SUMO, a stop duration, a departure deadline and a maximum overtime can be specified. PTV VISSIM's stop duration calculation has no correspondence in SUMO, since boarding and alighting rates cannot be modeled without pedestrian simulation. However, the mean value of the stop duration distribution can be kept consistent with the stop duration value of a bus stop in SUMO.

SUMO also allows private transport vehicles to perform stops. In PTV VISSIM, these vehicles can only interrupt their trips for a certain time in parking lots. The stop at a parking lot in PTV VISSIM is modeled by means of a parking routing decision, which can be modeled in SUMO with a rerouter to the corresponding parking lot. The probability of the rerouter corresponds to the parking rate of the routing decision in PTV VISSIM. Since parking rerouters provide a more appropriate equivalent to the parking routing decisions in PTV VISSIM than the SUMO's stops, these stops should not be used for private transport vehicles in SUMO. Therefore, they do not need to be considered by the consistency mechanisms.

## 4.10 Comparing Measurements

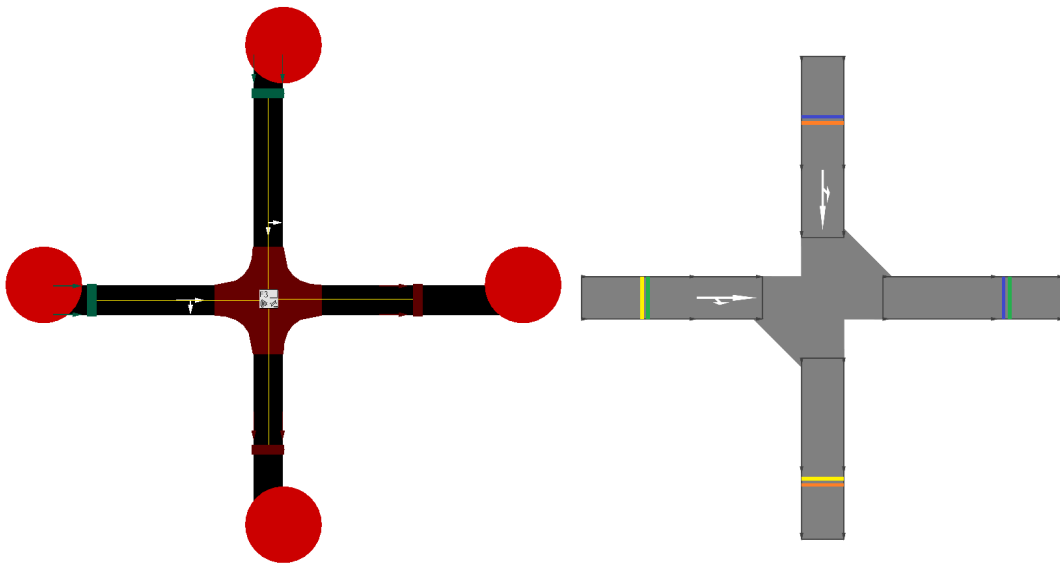
SUMO provides four different kinds of detectors that can be used to produce output data during the simulation. Induction loops and instant induction loops are used for local measurements to record data about vehicles passing a specific point in a network. Lane area detectors can be used to detect standing vehicle queues. Finally, entry-exit detectors detect the travel times and delays of vehicles. The travel time of a vehicle starts when it passes one of the detector's entries and ends when it leaves the observed area via one of the detector's exits.

PTV VISSIM has a large number of output options. Some of them correspond to the SUMO detectors described above. Data collection points can be used for local measurements, queue counters can be used to detect vehicle queues and travel time measurements can be used to measure travel times. Furthermore, these travel time measurements can be grouped in delay measurements to record vehicle delays.

SUMO's induction loops and instant induction loops correspond to data collection points in PTV VISSIM. Each data collection point in PTV VISSIM must also be contained in a data collection measurement, otherwise, PTV VISSIM will not record any data at that point.

Queue counters in PTV VISSIM correspond to lane area detectors in SUMO. However, queue counters do not cover an area in which vehicle queues are detected like lane area detectors do. Nevertheless, PTV VISSIM's evaluation configuration defines a maximum length up to which vehicle queues are detected. A queue counters position corresponds to the end position of a lane area detector and the maximum queue length equals the length of a lane area detector. PTV VISSIM's queue begin speed, below which a vehicle is considered as part of a vehicle queue is equal to the speed threshold of a lane area detector. PTV VISSIM also defines a queue end speed. This distinction is not possible in SUMO. To get a similar evaluation in PTV VISSIM and SUMO, the queue end speed must equal the begin speed. Alternatively, the queue end speed attribute can be seen as an individual feature of PTV VISSIM which motivates switching between both tools. In that case, the queue end speed of PTV VISSIM can simply be ignored by the consistency mechanisms. The maximum queue headway in PTV VISSIM corresponds to the jam threshold of a lane area detector. The time threshold of a lane area detector has no correspondence in PTV VISSIM which is why it should either default to 0 or again be seen as an individual feature of SUMO.

The last kind of detector in SUMO - entry-exit detectors - corresponds to travel time measurements and delay measurements in PTV VISSIM. Figure 4.10 shows an exemplary entry-exit detector on the left and the four corresponding travel time measurements in PTV VISSIM on the right. In the example, vehicles can either enter the intersection from the left or the top link and leave it via the right or the bottom link resulting in four possible routes on which the travel time has to be measured. PTV VISSIM's travel time measurements



**Figure 4.10:** Travel time measurement in SUMO with two entry-points (green) and two exit-points (red) on the left and the corresponding travel time measurements in PTV VISSIM on the right. For the travel time measurements to be distinguishable, they were shifted and highlighted in different colors.

always measure travel times between two points, hence four travel time measurements are required in the PTV VISSIM model. This demonstrates that a single entry- or exit-points in SUMO can correspond to the start- or end-points of multiple travel time measurements. So each combination of entry- and exit-points of one detector corresponds to one travel time measurement in PTV VISSIM. All travel time measurements corresponding to one entry-exit detector are grouped into the same delay measurement.

PTV VISSIM's evaluation configuration defines some global properties for each type of measurements, such as the time interval with which data should be aggregated or the queue counter properties. These properties then apply to all measurements of one type. In SUMO the aggregation intervals of the detectors (freq) can be modeled individually for each detector. As a solution, the attribute values of all detectors can either be aggregated to single values in PTV VISSIM or must be equal for all detectors. In the former case, it is not possible to propagate changes of these attributes in PTV VISSIM back to SUMO. In the latter case, SUMO's modeling power is restricted. However, the simulation results are more comparable in the latter case.

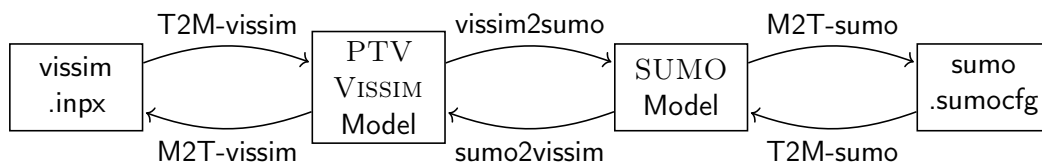




## 5 Implementation

In chapter 4 we developed relations between PTV VISSIM's and SUMO's metamodels. In some cases, we presented multiple options for keeping specific aspects of both models consistent. In this chapter, we will describe an implementation of Model-To-Model transformations between PTV VISSIM and SUMO models using the VITRUVIUS framework. As described in section 2.1.6, VITRUVIUS allows to define reactions to changes in a model as well as routines which restore the consistency after that change. Since not all model elements or attributes have a correspondence in the other model, it is important to propagate incremental changes. Values specified by the user would be lost if the entire network would be regenerated upon every change. Propagating incremental changes instead allows to only modify corresponding elements while keeping the custom values of unrelated model elements.

VITRUVIUS works with instances of the PTV VISSIM and SUMO metamodels in the XMI<sup>1</sup> format. However, the PTV VISSIM and SUMO editors require the files to have a different format. That format can be viewed as alternative XML<sup>2</sup>-based (textual) concrete syntax of the metamodels. PTV VISSIM models are persisted in the *inpx* format while SUMO uses *net*, *route* and *additional* files which are referenced by a single *sumocfg* file. Figure 5.1 shows the two formats of both PTV VISSIM and SUMO models as well as the implemented transformations between them.



**Figure 5.1:** The four representations of PTV VISSIM and SUMO models and the implemented transformations between them.

When changes are performed in one of the editors, the changed XML file has to be transformed into an instance of the corresponding metamodel (Text-To-Model) so that VITRUVIUS can perform the appropriate consistency preservation rules. After the consistency has been restored, the new consistent versions of the instances have to be transformed to the concrete syntax of their corresponding simulation programs (Model-To-Text), so that the changes become visible in both editors.

---

<sup>1</sup>XML Metadata Interchange

<sup>2</sup>Extensible Markup Language

The T2M<sup>1</sup> and M2T<sup>2</sup> transformations are syntactical horizontal transformations. The developed M2M<sup>3</sup> transformations are semantic transformations since they consider the semantics of the various model elements. Furthermore, they are horizontal transformations since PTV VISSIM and SUMO models have a similar level of abstraction. All three of them are exogenous transformations since they have different source- and target-metamodels respectively.

The transformation from the concrete textual syntax to metamodel instances (T2M transformation) is described in section 5.1, the M2T transformations are described in section 5.2 and the VITRUVIUS reactions are described in section 5.3.

## 5.1 Text to Model Transformations

For the implementation of transformations from the textual XML-based syntax to a metamodel instance we use the Java dialect Xtend [48] which is considered more readable and expressive than Java and offers several features to support code generation (see section 5.2). Since the input files of both editors have an XML-based format, an XML API can be used to parse the files. The two most common XML APIs (available for Java) are the de-facto standard SAX<sup>4</sup> [49] and the DOM<sup>5</sup> standard [50].

SAX has the disadvantage, that XML files cannot be validated before parsing them. However, since the XML files are generated by PTV VISSIM and SUMO, we can expect them to be valid XML documents. Furthermore, SAX parses the XML elements of a document sequentially and allows to define callback methods which are called at the beginning and end of every element. In this way, less memory is required compared to a DOM parser since only the current element has to be stored instead of the complete document. However, this requires to keep track of the object hierarchy in a separate data structure. This is necessary, because a SAX parser forgets all previous elements, as soon as the next element is encountered. The DOM API provides methods to modify the parsed document. However, these features are not needed when the contents of the XML document are merely transformed but not modified.

Independent of the used API, all elements of an XML document have to be handled exactly once during the transformation. The SAX API provides an order of those elements by nature, namely the order in which they appear in a file which constitutes a depth first search pre-order. Moreover, especially PTV VISSIM files are usually rather long, which means that only saving a single element at a time saves a significant amount of memory. Therefore, we use a SAX parser to transform PTV VISSIM and SUMO files into metamodel instances.

---

<sup>1</sup>Text-To-Model

<sup>2</sup>Model-To-Text

<sup>3</sup>Model-To-Model

<sup>4</sup>Simple API for XML

<sup>5</sup>Document Object Model

In the following, we will describe the implementation of the SAX callback methods which trigger the transformation of parsed XML elements.

When SAX parses the beginning of a new XML element, the method *startElement* of the registered *Handler* is called. The type of the parsed element is given in *qName* and the element's attributes are given in *attributes*. The given type can be used to select the appropriate method which transforms the XML element and its attributes into an element of the corresponding metamodel. Listing 5.1 shows an excerpt of the *startElement* method with two exemplary element types and their corresponding *handle*-methods.

---

```

override startElement(String qName, Attributes attributes) {
    var EObject eObj = null
    //handle the xml element depending on its type

    switch (qName) {
        case NETWORK: eObj = handleNet(attributes)
        case LINK: eObj = handleLink(attributes)
        ...
    }

    this.parent.push(eObj) //push the new object to the stack
}

```

---

**Listing 5.1:** SAX callback method *startElement*

Additionally to attributes, the parsed elements can contain references. In consequence, some elements are referenced, that have not been parsed yet. The missing reference can be resolved as soon as the relevant element is parsed. For every missing reference a *Job* object is created which holds the type of the missing object and its id as well as the object with the missing reference and the name of that reference. Listing 5.2 shows an exemplary *Job* for the missing reference *"lane"* of the given *parkingLot* which should reference a *LANE* with the id 42. Every time an element is finished by a call of the *endElement* method, all the *Jobs* that are still open can be checked whether they reference the current object. If they do so, the corresponding references can be resolved and the *Jobs* are removed as shown in listing 5.3.

---

```

new Job(LANE, parkingLot, "lane", 42)

```

---

**Listing 5.2:** An exemplary *Job* to resolve cross references

As mentioned above, the object hierarchy is persisted in an additional data structure, so that new objects can be assigned as childs of the appropriate parent object. To do so, a stack can be used which always holds the parent of the object that is currently parsed. After an XML element has been parsed and the corresponding model element has been created, that model element is pushed onto the stack as the new current parent before its children are being parsed. Once the parse reaches the end of an XML element the *endElement* method is called (see listing 5.3). Since the element and all its children have been parsed, it is removed from the stack.

The XML representations of PTV VISSIM models and especially SUMO models contain multiple options to represent some bits of information which requires special attention in the T2M transformations. During the reverse engineering process (see chapter 3) these options were reduced to a single representation in the respective metamodel to avoid redundancies. Since the XML files produced by the editors can contain either of the possible representations, the T2M transformations have to handle each of them and translate it to the single representation used in the metamodel.

For example, SUMO allows to either specify the speed limit for each lane individually or for the entire edge. In the SUMO metamodel, the speed limit is an attribute of lanes instead of edges. This can be handled, by setting the speed attribute of lanes to their edge's speed limit if the edge specifies it in the XML format. The Jobs mentioned above can be used to set the speed of all lanes, once an edge has been fully parsed.

---

```
override void endElement(..., String qName) {
    //Get the current object from the top of the stack
    val current = this.parent.peek()
    ...

    //Handle all late jobs of the type qName
    var jobs = new ArrayList<LateJob>()
    jobs.addAll(this.jobs.filter[it.xmlType == qName])

    for (job : jobs) {
        if (handleLate(current, job.obj, job.feature, job.value)) {
            this.lateJobs.remove(job) //Remove handled jobs
        }
    }

    ...
    this.parent.pop() //Remove current object from the stack
}
```

---

**Listing 5.3:** SAX callback method *endElement*

## 5.2 Model to Text Transformations

Changes to an instance of the PTV VISSIM or SUMO metamodel can be made visible in the editors by generating the textual XML-based representation of that instance. The Xtend language provides template expressions which allow to define a textual template that can be filled with attribute values of a metamodel instance at specific points. Listing 5.4 shows an example template which generate the XML representation of a `Link`. The `generateLink` method returns XML code as a `String`. Guillemet parentheses can be used to fill the template with dynamic values. In the example template, the `link`'s name and no are inserted as attributes of the XML element `link`. Xtend also allows to define conditionals and loops inside a template which simplifies generating sets of elements.

---

```
def static generateLink(Link link) '''
<link name="«link.name»" no="«link.no»" ...>
  <geometry>
    <linkPolyPts>
      «FOR p:link.linkPolyPts»
        <linkPolyPoint x="«p.x»" y="«p.y»" zOffset="«p.z»"/>
      «ENDFOR»
    </linkPolyPts>
  </geometry>
  <lanes>
    «FOR l:link.lanes»
      «generateLane(link, l)»
    «ENDFOR»
  </lanes>
</link>
'''
```

---

**Listing 5.4:** Xtend-template for the XML representation of a Link

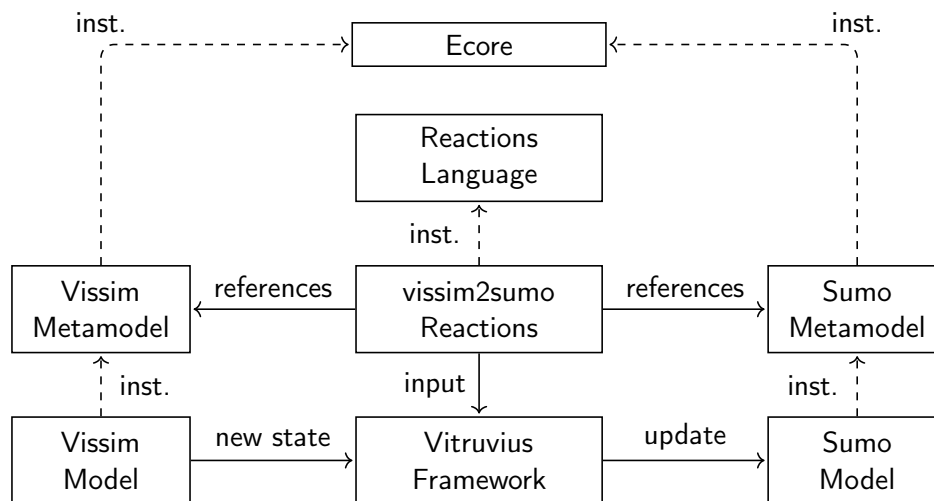
Since the PTV VISSIM and SUMO metamodels do not represent the entire domain available in their respective editors, some information that is required in the XML representations is not contained in the metamodels. Especially PTV VISSIM contains many attributes that are not part of the metamodel. Therefore, default values are assigned when performing M2T transformations. However, the documentation of SUMO does not provide default values for all attributes. These default values must chosen carefully, to prevent unexpected behavior during the simulation which could increases the effort to set them manually after the transformation.

### 5.3 Reactions

The VITRUVIUS framework provides the *Reactions* language which allows to define reactions to incremental changes which trigger routines to restore the consistency of the corresponding models.

Reactions form the core of the implemented consistency preservation mechanisms. They perform the actual translation of PTV VISSIM elements to SUMO elements and vice versa. They are divided into two groups: Reactions to changes in a PTV VISSIM model which restore the consistency in the corresponding SUMO model (*vissim2sumo*) and reactions to changes a SUMO model that execute updates in a PTV VISSIM model (*sumo2vissim*).

Figure 5.2 shows the overview on model transformations from section 2.1.4 with concrete instantiations of the involved elements. It shows a PTV VISSIM model as source model and a SUMO model as target model. Both are instances of their respective metamodels which in turn are instances of the meta-metamodel *Ecore*. The *vissim2sumo* reactions instantiate the *Reactions* language and are used by the VITRUVIUS framework as transformation engine in this process. The *sumo2vissim* reactions work analogously with SUMO as source model and PTV VISSIM as target model.



**Figure 5.2:** The components involved in a M2M transformation between PTV VISSIM and SUMO based on figure 2.3.

In this thesis, prototypes of these M2M transformations have been developed. They contain reactions to handle the creation and deletion of most model elements in both metamodels as well as changes to their attributes. It focuses on on the translation of the network geometry and stop infrastructures, vehicle types, vehicle inputs and flows as well as detectors, sensors and measurements to create output files. It provides transformations for conflict areas in PTV VISSIM to requests in SUMO as well as stop signs as exemplary transformations of traffic rules at intersections. Furthermore, transformations between PTV VISSIM's routing

decisions and SUMO's rerouters are only supported for static routes in PTV VISSIM and route-probability-rerouters in SUMO. Public transport and signal programs are not yet supported.

These reactions are designed to keep models consistent that are being built from scratch. However, it is also possible to integrate a single existing legacy model of PTV VISSIM or SUMO. Therefore, the creation of the existing model can be simulated to produce a change sequence that VITRUVIUS can use to generate the corresponding other model by executing the appropriate reactions. This constitutes a reconstructive integration strategy [28]. Furthermore, a linking integration strategy [28] could be applied, which builds correspondences between two existing models. This strategy is not investigated for this thesis, however, it could be studied in potential future work.

Some elements in a PTV VISSIM model have no correspondence in SUMO and vice versa. For example, links in PTV VISSIM have mandatory link behavior types which have no correspondence in SUMO. When a new edge is created in a SUMO model, the triggered transformation routine creates a corresponding link and sets that link's behavior type. To do so, heuristics are applied to select or create a fitting link behavior type. If the user wants to use a different type, he has to modify it manually afterward. Alternatively, the user can be asked to select the type for new links explicitly when they are created. VITRUVIUS provides a `userInteractor`, that can be used to prompt the user to select options from a list, confirm a notification or provide attribute values.

The prototype transformations implemented for this thesis contain two example user interaction: Selecting a link behavior type for links and connectors when transforming edges and connections from SUMO to PTV VISSIM and selecting the induction loop type when transforming data collection points from PTV VISSIM to SUMO.

In future work, these user interactions could be extended to cover more aspects of the two metamodels. However, too many user interactions could reduce the user's efficiency. Therefore, user interactions should be designed carefully, always considering the trade-off between the required effort during the transformation and the required effort to set the respective properties manually.





## 6 Evaluation

In chapter 5 we discussed the implementation of consistency preservation mechanisms in VITRUVIUS. These mechanism support users to keep PTV VISSIM and SUMO models consistent and are supposed to increase their efficiency and reduce the amount of inconsistencies. In this chapter, we will evaluate the developed consistency mechanisms to estimate their effect on the user's performance. Therefore, a study was conducted in which the participants were tasked to create consistent models with and without the developed consistency mechanisms. The study conducted in the course of this these was designed to provide a rough estimate of the consistency mechanism's impact. However, a profound evaluation should be conducted if the development of consistency mechanisms is continued.

### 6.1 Research Question and Hypotheses

The driving question behind the conducted study reads as follows:

**How does the use of the developed consistency mechanisms affect the efficiency and effectiveness of the user as well as the overall user experience and system usability?**

Keeping models consistent manually is a time consuming and error-prone process. Hence, using the automatic consistency mechanisms should reduce the time required to create consistent models and therefore increase the efficiency. Furthermore, it should reduce the amount of inconsistencies between both models which in turn increases the effectiveness. Since the consistency mechanisms restrict the set of valid PTV VISSIM and SUMO models, the usability of both tools is restricted as well which cold have a negative impact on the system usability and overall user experience of participants that are used to PTV VISSIM or SUMO. The developed consistency mechanisms allow to switch between PTV VISSIM and SUMO at any time during the modeling process to create different parts of the model explicitly with one of the two tools. On the one hand, the time overhead for switching between the tools could negatively affect the efficiency and system usability. On the other hand, once a user knows which tool to use best for certain parts of the model, the ability to switch between PTV VISSIM and SUMO should increase the efficiency and could have a positive impact on the overall user experience.

## 6.2 Methods

The efficiency of users can be determined by measuring the time they need to create consistent models. Additionally, the time required upfront to develop the consistency mechanisms should be measured as well. With this, the minimum amount of time can be determined after which the development and usage of these consistency mechanisms result in a higher efficiency than keeping models consistent manually. Furthermore, efficiency is one of the aspects that can be measured by a UEQ<sup>1</sup> [51].

Measuring the effectiveness of a user is not as simple as measuring their efficiency. Since the goal is to reduce inconsistencies between two models, the absolute number of remaining inconsistencies would be a suitable indicator for effectiveness. However, counting inconsistencies is not a trivial task and would require a formal specification of inconsistency. As an alternative, the simulation results of the PTV VISSIM and SUMO models can be compared. Assuming that consistent models produce more similar results than inconsistent ones, the difference between two simulation results can be used as an indicator for effectiveness.

The consistency of two models can be estimated by aggregating the simulation results for each of the following measurements (based on the modeling task in figure A.11) into a single value per model: average vehicle speed  $\overline{v_m}$ , vehicle count in the main stream  $q_{m,1}$ , vehicle count in the merging stream  $q_{m,2}$ , travel time in the main stream  $\overline{t_{m,1}}$  and travel time in the merging stream  $\overline{t_{m,2}}$ . Since these values are measured in different units, they have to be normalized before they can be summed up to a single value. The normalization factors for each measurement can be derived from the modeling task used during the study.

The given speed limit of  $120 \frac{km}{h}$  can be used to normalize the measured average speed  $\overline{v_m}$  of a model  $m$ . The specified flow values of  $2000 \frac{Veh}{h}$  in the main stream and  $1000 \frac{Veh}{h}$  in the merging stream can be used to normalize the respective measured flows  $q_{m,1}$  and  $q_{m,2}$ . Travel times are measured over a distance of  $900 m$  for the main stream and  $724 m$  for the merging stream. At a speed of  $120 \frac{km}{h}$ , this results in optimal travel times of  $27 s$  for vehicles in the main stream and  $21.72 s$  in the merging stream. These optimal travel times can be used to normalize the measured travel times  $\overline{t_{m,1}}$  and  $\overline{t_{m,2}}$  respectively.

To determine the consistency of two models  $a$  and  $b$ , their normalized measurements can be compared pairwise. The absolute difference for every pair of measurements can be aggregated by a weighted sum into a single consistency value as shown in (6.1).

$$(6.1) \quad \delta_{a,b} = \frac{1}{5} \left( \frac{|\overline{v_a} - \overline{v_b}| \left[ \frac{km}{h} \right]}{120 \left[ \frac{km}{h} \right]} + \frac{|q_{a,1} - q_{b,1}| \left[ \frac{Veh}{h} \right]}{2000 \left[ \frac{Veh}{h} \right]} + \frac{|q_{a,2} - q_{b,2}| \left[ \frac{Veh}{h} \right]}{1000 \left[ \frac{Veh}{h} \right]} + \frac{|\overline{t_{a,1}} - \overline{t_{b,1}}| [s]}{27 [s]} + \frac{|\overline{t_{a,2}} - \overline{t_{b,2}}| [s]}{21.72 [s]} \right)$$

---

<sup>1</sup>User Experience Questionnaire

A consistency value of zero indicates perfect consistency of  $a$  and  $b$  in terms of this metric. For this study, we will consider each of the measurements equally important which is why all normalized measurements receive the same weight of  $\frac{1}{5}$ . Let  $exp$  be the expected model that produces measurement results which are equal to their respective normalization values and therefore matches the task's description. Hence, the consistency  $\delta_{exp,m}$  of a model  $m$  and  $exp$  describes how well  $m$  models the task. In the following we will write  $\delta_v$  for  $\delta_{exp,v}$ ,  $v$  being a participants PTV VISSIM model and  $\delta_s$  for  $\delta_{exp,s}$ ,  $s$  being a participants SUMO model.

The system usability of PTV VISSIM and SUMO can be assessed using the System Usability Scale [52] which is designed to compare the usability of arbitrary systems (see figure A.13). The SUS<sup>1</sup> usually consists of a ten-item Likert scale [53]. Each of those ten items (questions) can be rated on a scale of five options ranging from 'Strongly Agree' to 'Strongly Disagree'. Each option is assigned a value from 0 to 4 so that the options selected by a participant can be summed up to a single value on a scale of 0 – 40. This aggregated value is scaled with a factor of 2.5 which results in the SUS score on a scale of 0 – 100.

Furthermore, a User Experience Questionnaire [51] (see figure A.12) can be used to measure the user experience in six categories: attractiveness, perspicuity, efficiency, dependability, stimulation and novelty. The questionnaire consists of 26 items, each rated on a scale of seven options. The rating scales range between two terms with opposite meanings like 'attractive' and 'unattractive', depending on the item. The seven rating options are assigned values from -3 to +3, -3 for the most negative and +3 for the most positive answer. The values of all 26 items are aggregated into six scores for the six categories mentioned above as well as a total user experience score each on a scale from -3 to +3.

## 6.3 Sample

The user group targeted by PTV VISSIM and SUMO consists of traffic engineers which use the tools for project work or research. The participants considered in this study were students of transport studies at the Karlsruhe Institute of Technology (KIT) which are potential future traffic engineers or scientists. The participants selected for this study had experience in using at least on of the two tools. Nielsen [54] suggests that on average four to five participants identify 70% to 80% of usability problems. Since there are two groups in the first session, each group should have the same number of participants. Therefore, a sample of four participants was used in this study.

---

<sup>1</sup>System Usability Scale

## 6.4 Procedure

The study is divided into two sessions for each participant. The first session is used to measure base values when only using PTV VISSIM and SUMO without VITRUVIUS. During the second session, the participants use the VITRUVIUS consistency mechanisms while modeling. The data measured in the second session can then be compared to the base values from the first session to estimate the impact of the consistency mechanisms.

To ensure that all participants share a minimum knowledge base concerning PTV VISSIM and SUMO, they receive an introduction to both tools that includes all features required to complete the task (see figures A.1 to A.5 and figures A.6 to A.10).

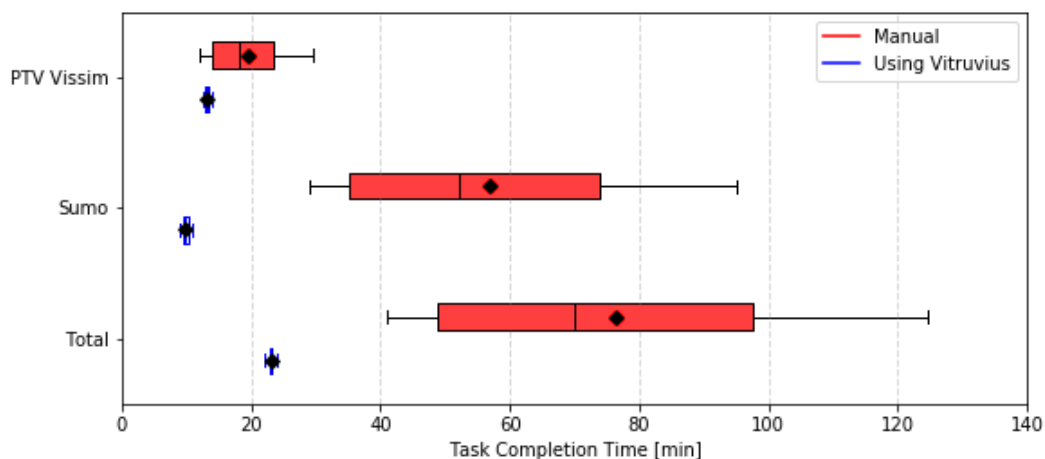
During the first session, each participant is tasked to model a motorway section with an onramp using both PTV VISSIM and SUMO (see figure A.11). The files created during each session are logged in the background using a script to automatically commit changes to a git repository. This automatically generated log is used to determine the time the participants spend on creating the two models. The time between the first and the last modification of the model files equals the task completion time. Since the participants have to model the same task twice during the first session, once with PTV VISSIM and once with SUMO, learning effects could arise as the participants become more familiar with the task after having completed it once. To overcome these learning effects, half of the participants start with PTV VISSIM while the other half starts with SUMO. Each time they complete the task with one of the tools, the participants are asked to answer a questionnaire which consists of a UEQ and a SUS. This produces user experience and system usability ratings for both PTV VISSIM and SUMO.

In the second session, the participants are asked to model the same task as in the first session. In a more profound, future study, half of the participants should start with session one and the other half should start with session two to overcome learning effects. However, in this thesis all participants started with session one. To reduce possible learning effects, the sessions one and two took place two months apart from each other. In the second session, the task is modeled only once. This time, the participants use both the PTV VISSIM and the SUMO editor. The SUMO editor is used to model the motorway's geometry and to set the speed limit while vehicle flows and measurements are modeled in PTV VISSIM. For the study, a simple demo application was created, which starts a batch transformation for a selected model of one simulation program to generate a corresponding model for the other program. A simple graphical user interface allows the participants to trigger the transformation in either direction explicitly. Similar to the first session, the participants are asked to answer a questionnaire after completing the task. However, this time the system usability and user experience is measured for the combined interaction with PTV VISSIM and SUMO and VITRUVIUS in the background.

## 6.5 Results

In this section, we will present the results of the study described in section 6.4. We will use box-plots to present the data collected during the study with the following notation: Each box-plot consists of two whiskers which mark the minimum and the maximum value and a colored box marking the range between the lower and upper quartile. The median value is marked by a black line within the colored box and the mean value is marked by a black diamond shape.

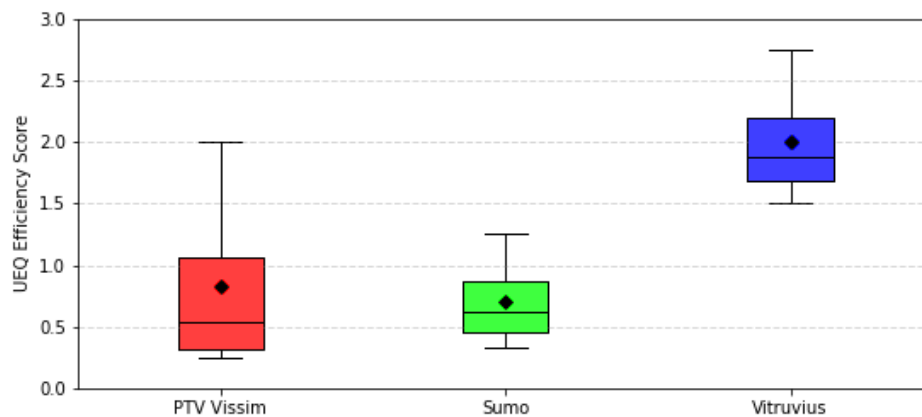
During the first session, the participants have to create two models manually. In the second session they only create one model manually since the other model is generated automatically. Therefore, a reduction of the task completion times to a half can be expected. Moreover, the measured task completion times in figure 6.1 show an even greater reduction. The task completion times in the first session range from 12 to 30 minutes for PTV VISSIM and from 30 to 89 minutes for SUMO which results in total task completion times ranging from 40 to 124 minutes. The task completion times of the second session can be divided into two portions, one for each tool. Especially time spent with the SUMO editor is significantly smaller than in the first session. A possible cause for this reduction is that the SUMO editor is only used to model the networks geometry, which is properly supported while other aspects that are more complicated to model in SUMO are modeled in PTV VISSIM. This is also a possible cause for the smaller variation of the task completion times in the second session. In total, the average task completion time is reduced from 76 minutes in the first session to 23 minutes in the second session which constitutes a reduction by 69.7% or 53 minutes on average.



**Figure 6.1:** The impact of consistency mechanisms on a participant's task completion time. The task completion time for the manually created models are given in red and for models generated by VITRUVIUS in blue

The time required to develop the consistency mechanisms can be estimated similarly to the task completion times by automatically logging changes every minute. This produces an estimate of 137.5 hours of development time. The evaluation of task completion times shows that on average 42 minutes can be saved per hour by using the developed consistency mechanisms. This results in an amortization time of 196.5 hours of modeling, after which the consistency mechanisms have saved more time than they required to develop.

Additionally to the measured objective task completion times, the UEQ can be evaluated to obtain the participants' subjective rating of the experienced efficiency. Figure 6.2 shows the UEQ-efficiency score for PTV VISSIM and SUMO in the first session which are 0.8 and 0.7 on average. The UEQ-efficiency score in the second session rates the efficiency of the entire interaction with PTV VISSIM and SUMO and VITRUVIUS in the background and evaluates to a mean score of 2.0. This increase in UEQ-efficiency ratings matches the measured decrease in task completion times.

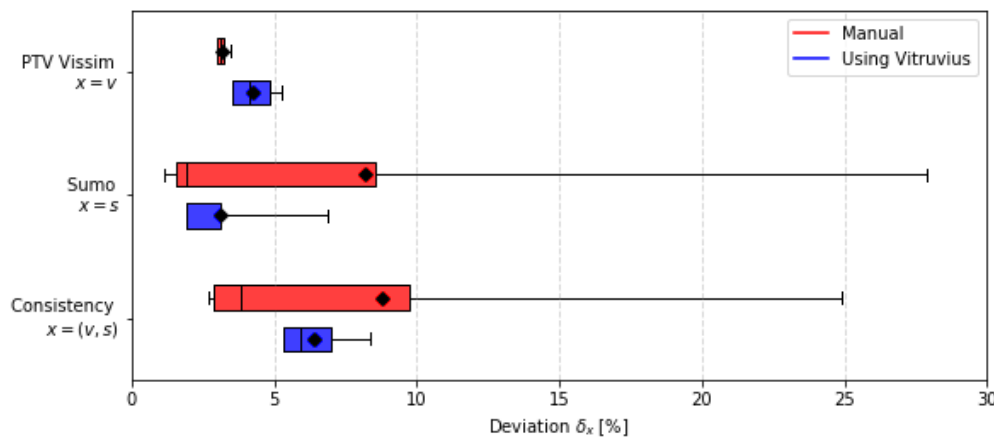


**Figure 6.2:** The impact of consistency mechanisms on a participant's subjective UEQ rating of efficiency. PTV VISSIM (red) and SUMO (green) were rated separately in the first session. The combined interaction was rated during the second session (blue).

The developed consistency mechanisms result in a significantly higher efficiency. However, the gained efficiency is only useful if the produced models are consistent. Figure 6.3 shows how well the created PTV VISSIM and SUMO models match the expected model as well as the consistency of both models using the metric described in section 6.2. The PTV VISSIM models generated during the first session have a 3.1 % deviation from the expected model on average. The manually created SUMO models were on average less consistent, due to some modeling errors. Their deviation ranges from 1.1 % to 27.8 %.

The consistency of each participant's PTV VISSIM and SUMO model in the first session ranges from 2.7 % to 24.8 % with an average of 8.8 %. The PTV VISSIM models produced during the second session have a slightly larger deviation from the expected model with an average deviation of 4.2 %. The SUMO models instead have on average a smaller deviation (3.1 %) from the expected model than manually created ones. One possible reason for this

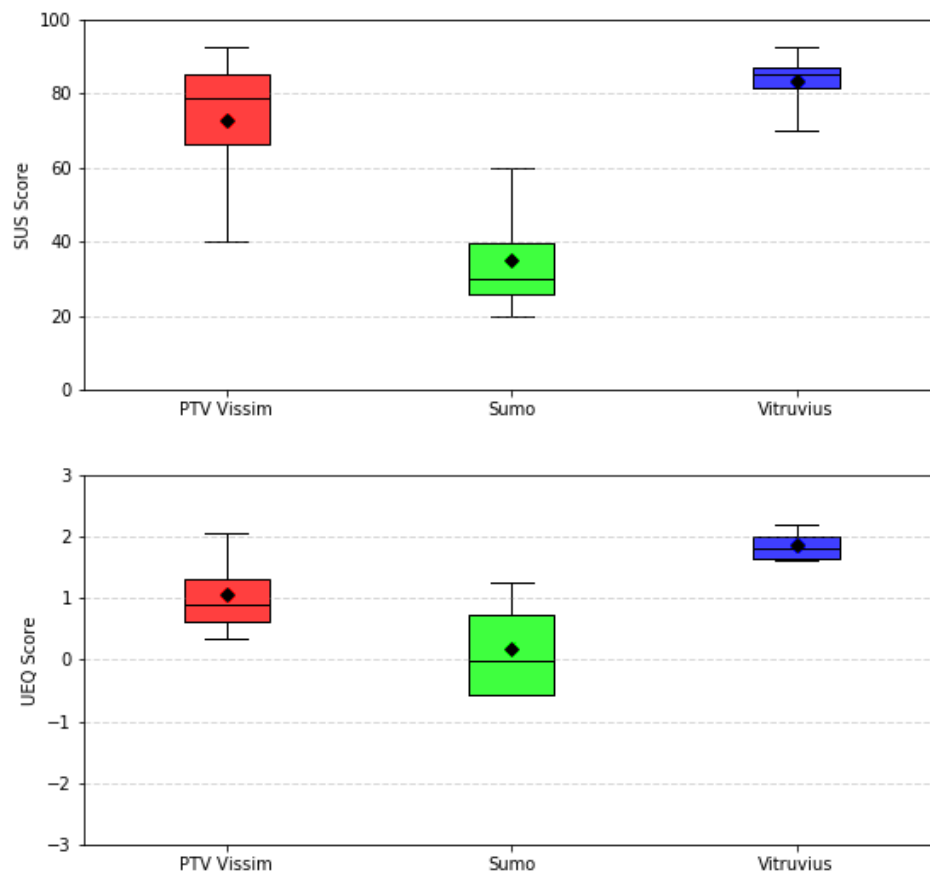
is that the SUMO editor was merely used to model the net geometry, while the rest was generated by VITRUVIUS. The consistency of the created models is slightly smaller in the second session with an average deviation of 6.4 %.



**Figure 6.3:** The impact of consistency mechanisms on the consistency of the produced models and the deviation of the PTV VISSIM and SUMO models from the expected model. The consistency of the manually created models are given in red and the models generated by VITRUVIUS in blue.

Interestingly, three out of four participants created identical SUMO networks in the second session which produce the same simulation results. Also, two out of four PTV VISSIM models were identical. So the remaining inconsistency between the PTV VISSIM and SUMO models can be further reduced by improving the transformations. A current source of inconsistency are missing default values for attributes in the SUMO metamodel which are not specified in the documentation of SUMO.

Finally, figure 6.4 shows the impact of the developed consistency mechanisms on the system's usability and the overall user experience. The PTV VISSIM editor was rated with an average SUS score of 72.5 while the SUMO editor merely received an average SUS score of 36.2. The interaction with both editors and VITRUVIUS in the background was rated even higher than PTV VISSIM with an average SUS score of 83.1. The UEQ scores show a similar relation with SUMO having the least average score (0.1) followed by PTV VISSIM with an average score of 1.0. The combined interactions received the highest average score of 1.8. These high scores as well as the participants' qualitative feedback indicate that being able to select the preferred editor for each sub-task is a useful property that could be enhanced in the future.



**Figure 6.4:** The impact of consistency mechanisms on the system usability (top) and the overall user experience (bottom). PTV VISSIM (red) and SUMO (green) were rated separately in the first session. The combined interaction was rated during the second session (blue).



## 7 Conclusion

In this chapter, we conclude this thesis by summarizing its results in section 7.1 and providing an outlook on possible future work in section 7.2.

### 7.1 Results

In this thesis, we developed consistency mechanisms for models of the microscopic traffic simulation programs PTV VISSIM and SUMO to support concurrent development of consistent models with both programs.

To evaluate the developed consistency mechanisms, we conducted a study to estimate the effect of the developed consistency mechanisms. To evaluate the user's efficiency, we compared the time they spent on creating consistent PTV VISSIM and SUMO models with and without the consistency preservation of VITRUVIUS. Additionally, we compared the simulation results of the created models to evaluate the user's effectiveness in creating consistent models. Therefore we proposed a simple metric to measure the consistency of two models based on these simulation results. Furthermore, a system usability and user experience questionnaire were used to estimate the impact of VITRUVIUS on the usability.

The evaluation showed, that the time required to create two consistent models could be reduced by 69.7%, while the produced models are slightly more consistent. Furthermore, the system usability was rated significantly higher compared to the SUMO editor and on average slightly higher than the PTV VISSIM editor. Similarly, the overall user experience of the combined interaction was rated higher than the individual interactions.

### 7.2 Outlook

In this thesis, we presented metamodels of PTV VISSIM and SUMO which contain the most important aspects. However, these metamodels could be extended in the future, for example, by including pedestrian simulation to keep more aspects of PTV VISSIM and SUMO models consistent. Moreover, other traffic simulation or traffic planning programs like AIMSUN or PTV VISUM could be integrated by creating additional metamodels and appropriate transformations.

While comparing PTV VISSIM's and SUMO's metamodels, we provided different options for keeping specific aspects consistent. However, for each choice, only one option was implemented in the form of a reaction in VITRUVIUS. Since we used an inductive definition of consistency through reactions and routines, the definition of consistent models can be made configurable. For example, if multiple heuristics are implemented for transforming the visibility at intersections, the user could choose, which of them should be applied during the transformation.

Finally, investigating the relation between PTV VISSIM's and SUMO's driving behavior is of interest. We decided not to transform driving behavior elements between PTV VISSIM and SUMO models since these are often used to calibrate the networks. Without a profound analysis of these elements, their semantics and their influence on the simulation results, both models should be calibrated individually. However, being able to transform a calibrated PTV VISSIM model into a SUMO model that is still calibrated could further reduce the effort required to create consistent models.

---

## Bibliography

- [1] *PTV Vissim 11 - User Manual*. PTV AG, Haid-und-Neu-Str. 15 76131 Karlsruhe Germany, September 2018. Vissim Version 11.00 - 12 [81870].
- [2] *Simulation of Urban MObility - Documentation*. German Aerospace Center (DLR), January 2020. URL [https://sumo.dlr.de/userdoc/SUMO\\_User\\_Documentation.html](https://sumo.dlr.de/userdoc/SUMO_User_Documentation.html).
- [3] Max E Kramer, Erik Burger, and Michael Langhammer. View-centric engineering with synchronized heterogeneous models. In *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, page 5. ACM, 2013.
- [4] Colin Atkinson and Thomas Kühne. Kühne, t.: Model-driven development: a meta-modeling foundation. *iee software* 20(5), 36-41. *Software, IEEE*, 20:36 – 41, 10 2003.
- [5] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973. URL <https://archive.org/details/Stachowiak1973AllgemeineModelltheorie>.
- [6] Leonhard Euler. *Solutio problematis ad geometriam situs pertinentis*. 1741.
- [7] Merian Erben. File:image-koenigsberg, map by merian-erben 1652.jpg — wikimedia commons, the free media repository. Wikipedia Commons, June 2016. URL <http://commons.wikimedia.org/w/index.php?curid=1447648>. [Gemeinfrei; Online; accessed 29-October-2019].
- [8] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2): 171–188, May 2005. ISSN 1619-1374.
- [9] Thomas Stahl, Markus Völter, Sven Efftinge, and Arno Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt-Verlag, 1. auflage edition, 2005. ISBN 3-89864-310-7.
- [10] David Harel and Bernhard Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, October 2004. ISSN 0018-9162.
- [11] Jos B Warmer and Anneke G Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.

- [12] *Meta Object Facility (MOF)*. Object Management Group (OMG), 2.5.1 edition, October 2016. URL <http://www.omg.org/spec/MOF/>.
- [13] Omg unified modeling language tm (omg uml), infrastructure. 2011. URL <https://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/>. Version 2.4.1.
- [14] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.
- [15] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 032119442X.
- [16] Jean Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.
- [17] Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. A taxonomy of model transformation. In *Proc. Dagstuhl Seminar on "Language Engineering for Model-Driven Software Development"*. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl. Electronic, 2005.
- [18] Object Management Group (OMG). Meta object facility (mof) 2.0 query/view/transformation (qvt), 2016. URL <https://www.omg.org/spec/QVT>.
- [19] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720, 2006.
- [20] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17–es, May 2007. ISSN 0164-0925.
- [21] Erik Burger, Jörg Henss, Martin Küster, Steffen Kruse, and Lucia Happe. View-based model-driven software development with modeljoin. *Software & Systems Modeling*, 15(2):473–496, 2016.
- [22] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From state- to delta-based bidirectional model transformations: The symmetric case. In *Model Driven Engineering Languages and Systems*, 2011.

- 
- [23] Systems and software engineering — architecture description, December 2011. URL <http://cabibbo.dia.uniroma3.it/asw/altrui/iso-iec-ieee-42010-2011.pdf>.
- [24] Erik Burger. *Flexible views for view-based model-driven development*, volume 15. KIT Scientific Publishing, 2014.
- [25] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. Orthographic software modeling: A practical approach to view-based development. In *Evaluation of Novel Approaches to Software Engineering*, pages 206–219. Springer, 2009. URL [https://link.springer.com/chapter/10.1007/978-3-642-14819-4\\_15](https://link.springer.com/chapter/10.1007/978-3-642-14819-4_15).
- [26] Max Emanuel Kramer. *Specification Languages for Preserving Consistency between Models of Different Languages*. PhD thesis, 2017.
- [27] Manar Mazkatli, Erik Burger, Jochen Quante, and Anne Koziolk. Integrating semantically-related legacy models in vitruvius. In *Proceedings of the 10th International Workshop on Modelling in Software Engineering*, pages 41–48, 2018.
- [28] Sven Leonhardt, Benjamin Hettwer, Johannes Hoor, and Michael Langhammer. Integration of existing software artifacts into a view-and change-driven development approach. In *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, pages 17–24, 2015.
- [29] Dieter Lohse and Werner Schnabel. *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung: Band 2-Verkehrsplanung*, volume 3. Beuth Verlag, 2011.
- [30] Werner Schnabel and Dieter Lohse. *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung: Band 1-Straßenverkehrstechnik*, volume 3. Beuth Verlag, 2011.
- [31] Martin Treiber and Arne Kesting. *Verkehrsdynamik und-simulation: Daten, Modelle und Anwendungen der Verkehrsflussdynamik*. Springer-Verlag, 2010.
- [32] Martin Fellendorf, Markus Friedrich, and Peter Vortisch. Kopplung makroskopischer und mikroskopischer verkehrsmodelle-ein verfahren für die integration von großräumiger planung und detailplanung. *Tagungsband der 18. Verkehrswissenschaftlichen Tage*, 2001.
- [33] OpenStreetMap contributors. Planet dump retrieved from <https://planet.osm.org> . <https://www.openstreetmap.org>, 2017.
- [34] Jaume Barceló et al. *Fundamentals of traffic simulation*, volume 145. Springer, 2010.

- [35] Rainer Wiedemann. *Simulation des Verkehrsflusses*. PhD thesis, IfV, 1974.
- [36] Umair Durrani and Chris Lee. Calibration and validation of psychophysical car-following model using driver's action points and perception thresholds. *Journal of Transportation Engineering, Part A: Systems*, 145(9):04019039, 2019.
- [37] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.
- [38] Christian Gawron. Simulation-based traffic assignment. 1998.
- [39] Stefan Krauß. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*. PhD thesis, Dt. Zentrum für Luft-und Raumfahrt eV, 1998.
- [40] Jakob Erdmann. Sumo's lane-changing model. In Michael Behrisch and Melanie Weber, editors, *2nd SUMO User Conference*, volume 13 of *Lecture Notes in Control and Information Sciences*, pages 105–123. Springer Verlag, 2015. URL <https://elib.dlr.de/102254/>.
- [41] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo—simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.
- [42] Extension tools to import features from vissim \*.inpx files. <https://sumo.dlr.de/docs/Tools/Import/VISSIM.html>, . Accessed: 2020-27-05.
- [43] Softeis. File:autobahn anschluss1.jpg — wikimedia commons, the free media repository. Wikipedia Commons, June 2005. URL <https://commons.wikimedia.org/w/index.php?curid=619458>. [Gemeinfrei; Online; accessed 01-April-2020].
- [44] *Vissim - COM*. PTV AG, Haid-und-Neu-Str. 15 76131 Karlsruhe Germany, 2015. Vissim Version 11.00 - 12 [81870].
- [45] Networks/plainxml - connection descriptions. [https://sumo.dlr.de/docs/Networks/PlainXML.html#connection\\_descriptions](https://sumo.dlr.de/docs/Networks/PlainXML.html#connection_descriptions), . Accessed: 2019-15-11.
- [46] Networks/sumo road networks - connections. [https://sumo.dlr.de/docs/Networks/SUMO\\_Road\\_Networks.html#connections](https://sumo.dlr.de/docs/Networks/SUMO_Road_Networks.html#connections), . Accessed: 2019-15-11.
- [47] FGSV. *Handbuch für die Bemessung von Straßenverkehrsanlagen*. Forschungsgesellschaft für Straßen- und Verkehrswesen - Kommission Bemessung von Straßenverkehrsanlagen, ausgabe 2015 edition, 2015.

- [48] Xtend - modernized java. URL <https://www.eclipse.org/xtend/>.
- [49] David Megginson and David Brownell. Simple api for xml (sax). URL <http://www.saxproject.org>.
- [50] Document object model (dom). URL <https://dom.spec.whatwg.org>.
- [51] Martin Schrepp. *User Experience Questionnaire Handbook*, 09 2015.
- [52] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [53] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.
- [54] Jakob Nielsen. Why you only need to test with 5 users, 2000.




















# A Appendix

## Studie zur konsistenten Modellierung mit PTV Vissim und SUMO – SUMO

### 1. Allgemeines

#### Editor-Modi:

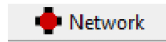
Modus	Icon	Zu finden unter
Knoten/Kanten-Modus		 Network
Verschieben-Modus		 Network
Inspektions-Modus		 Network  Demand
Additional-Modus		 Network
Verbinder-Modus		 Network
Routen-Modus		 Demand
Fahrzeuge-Modus		 Demand


**Speichern:** Das Netz, die Routen und Zuflüsse und die Additional-Objekte werden in unterschiedlichen Dateien gespeichert:

- Netz-Datei speichern: **File > Save Network as**, Dateiname mit Endung **„.net.xml“** angeben. Alternativ Tastenkombination **Strg + Shift + S** bzw. **Strg + S**.
- Routen-Datei speichern: **File > Demand elements > Save demand elements**, Dateiname mit Endung **„.rou.xml“** angeben. Alternativ Tastenkombination **Strg + Shift + D**.
- Additional-Datei speichern: **File > Additional + shapes > Save Additional**, Dateiname mit Endung **„.add.xml“** angeben. Alternativ Tastenkombination **Strg + Shift + A**.

**Figure A.1:** The introduction to SUMO used in the study given in German. It contains all editor modes required to finish the assigned task.

## 2. Netzaufbau mit SUMO (Network)



- 
  - **Knoten** (Junction) können im **Knoten/Kanten-Modus** durch **Linksklick** erzeugt werden.
    - 
      - **Position** (x, y) kann im **Verschieben-Modus** durch ‚**Drag-and-Drop**‘ oder im **Inspektions-Modus** geändert werden.
    - 
      - **Vorfahrtstyp** (type) kann im **Inspektions-Modus** geändert werden.
        - Priority: Vorfahrtsregelung, Standard Einstellung
        - Zipper: Zwei Fahrstreifen werden zu einem vereinigt
- 
  - **Kanten** (Edge) werden im **Knoten/Kanten Modus** zwischen zwei nacheinander eingefügten Knoten erzeugt.
    - 
      - **Geschwindigkeit** (speed) kann im **Inspektions-Modus** geändert werden.  
(**Achtung:** Geschwindigkeit in **m/s**)
    - 
      - **Fahrstreifen** (Lane) können im **Inspektions-Modus** per **Rechtsklick** auf eine Kante hinzugefügt werden (lane operations > Duplicate Lane).
        - **Breite** (width) kann im **Inspektions-Modus** geändert werden.
- 
  - **Verbinder** (Connections) können im **Verbinder-Modus** geändert werden. Nach dem Auswählen eines Quell-Fahrstreifens (**Linksklick**, dann hellblau) können Verbindungen durch **Linksklick** auf die gewünschten Ziel-Fahrstreifen erzeugt (dunkelgrün) oder gelöscht (hellgrün) werden (sofern keine Konflikte (gelb) existieren). Änderungen mit **OK** bestätigen. Zwei Verbinder die auf dem **selben Fahrstreifen** enden, können **nicht** im Editor erstellt werden. Diese können im **Texteditor (.net.xml Datei)** wie folgt modelliert werden:

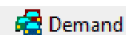
```
<junction id="gneJ3" type="zipper" ... incLanes="<IN>_0 <IN>_1 <IN>_2" ...>
  <request index="0" response="010" foes="010"/>
  <request index="1" response="001" foes="001"/>
  <request index="2" response="000" foes="000"/>
</junction>
...
<connection from="<IN>" to="<OUT>" fromLane="0" toLane="0" dir="s" state="Z"/>
<connection from="<IN>" to="<OUT>" fromLane="1" toLane="0" dir="s" state="Z"/>
<connection from="<IN>" to="<OUT>" fromLane="2" toLane="1" dir="s" state="M"/>
```










(Fehlenden Verbinder einfügen, vereinigte Verbinder erhalten Zustand ‚Z‘)

**Figure A.2:** The introduction to SUMO used in the study given in German. It contains all infrastructure objects required to finish the assigned task.

---

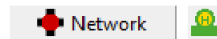
### 3. Zuflüsse und Routen in Sumo (Demand)



-  • **Routen** (Routes) können im **Routen-Modus** erzeugt werden. Dazu kann eine Folge von Kanten im Editor ausgewählt werden (**Linksklick**). Die Auswahl muss dann noch mit **‚Create Route‘** bestätigt werden.
-  • **Zuflüsse** (Vehicles – flow) können im **Fahrzeuge-Modus** durch **Linksklick** auf eine Route erzeugt werden. Zuvor können jedoch im **Attribut-Fenster** (links) die Eigenschaften des Zuflusses gewählt werden. Alternativ können diese Eigenschaften auch später im **Inspektions-Modus** geändert werden. Hier muss unter **‚Vehicle‘** der Typ **‚routeFlow‘** gewählt werden.
-   ○ **Fahrzeugtyp** (Vehicle Type) kann bei der Erzeugung des Zuflusses oder später im **Inspektions-Modus** geändert werden.
  - **DEFAULT\_VEHTYPE**: Standard Fahrzeugtyp für **PKW**
-   ○ **Anfangsgeschwindigkeit** (departSpeed) kann bei der Erzeugung des Zuflusses oder später im **Inspektions-Modus** geändert werden.  
(**Achtung**: Geschwindigkeit in **m/s**)
-   ○ **Fahrzeuge pro Stunde** (vehsPerHour) kann bei der Erzeugung des Zuflusses oder später im **Inspektions-Modus** geändert werden.
-  ○ **Wahrscheinlichkeit** (probability) kann **nur** bei der Erzeugung des Zuflusses und **nicht** später im **Inspektions-Modus** geändert werden. Dazu muss zunächst die Option **‚probability‘** ausgewählt werden.

**Figure A.3:** The introduction to SUMO used in the study given in German. It contains all demand objects required to finish the assigned task.

#### 4. Auswertung mit Sumo (Additionalns)



- **Messquerschnitte** (e1Detector) können im **Additionalns-Modus** erzeugt werden. Dazu muss unter ‚**Additional element**‘ die Option ‚**e1Detector**‘ gewählt werden. Ein Messquerschnitt wird durch **Linksklick** auf eine Kante an der gewünschten Stelle eingefügt. Je Fahrstreifen sollte ein Messquerschnitt erzeugt werden.



- **Position** (pos) kann bei der Erzeugung des Messquerschnitts oder später im **Verschieben-Modus** durch ‚**Drag-and-Drop**‘ oder im **Inspektions-Modus** geändert werden.



- **Reise-/Verlustzeitmessungen** (e3Detector) können im **Additionalns-Modus** erzeugt werden. Dazu muss unter ‚**Additional element**‘ die Option ‚**e3Detector**‘ gewählt werden. Eine Messung wird durch **Linksklick** an einer beliebigen Stelle eingefügt.



- **Intervall** (freq) kann bei der Erzeugung der Messung oder später im **Inspektions-Modus** geändert werden.



- **Anfangspositionen/Endpositionen** (detEntry/detExit) können im **Additionalns-Modus** erzeugt werden. Dazu muss unter ‚**Additional element**‘ die Option ‚**detEntry**‘ bzw. ‚**detExit**‘ gewählt werden. Vor dem Einfügen muss ein e3Detektor ausgewählt werden, zu dem der neue Anfangs-/Endpunkt gehören soll. Eine Anfangs-/Endposition wird durch **Linksklick** auf eine Kante an der gewünschten Stelle eingefügt. Bei Kanten mit mehreren Fahrstreifen muss je Fahrstreifen eine Anfangsposition erzeugt werden.



- **Position** (pos) kann bei der Erzeugung des Anfangs-/Endpunktes oder später im **Verschieben-Modus** durch ‚**Drag-and-Drop**‘ oder im **Inspektions-Modus** geändert werden.

**Figure A.4:** The introduction to SUMO used in the study given in German. It contains all evaluation objects required to finish the assigned task.

---

## 5. Simulation

- Netz, Routen und Additional-Objekte werden in verschiedenen Dateien gespeichert. Um diese drei Dateien zu einer Simulation zusammen zu fassen, muss eine **.sumocfg**-Datei erstellt werden. Diese beinhaltet folgende Daten:
  - Netz-Datei (net-file)
  - Routen-Datei (route-files)
  - Additional-Datei (additional-files)
  - Begin der Simulation (begin)
  - Ende der Simulation (end)

Diese Daten können in einem **Texteditor (.sumocfg Datei)** wie folgt angegeben werden:

```
<configuration>
  <input>
    <net-file value="<NETZ-DATEI>" />
    <route-files value="<ROUTEN-DATEI>" />
    <additional-files value="<ADDITIONAL-DATEI>" />
  </input>
  <time>
    <begin value="0" />
    <end value="3600" />
  </time>
</configuration>
```






Diese Konfiguration muss als Datei mit der Endung **„.sumocfg“** gespeichert werden.

**Figure A.5:** The introduction to SUMO used in the study given in German. It describes the structure of sumocfg files which is required to finish the assigned task.

## Studie zur konsistenten Modellierung mit PTV Vissim und SUMO – PTV Vissim

### 1. Allgemein

**Editor-Modi:**

Modus	Icon
Strecken-Modus	
Fahrzeugzuflüsse-Modus	
Fahrzeugrouten-Modus	
Messquerschnitte-Modus	
Fahrzeugreisezeiten-Modus	

**Figure A.6:** The introduction to PTV VISSIM used in the study given in German. It contains all editor modes required to finish the assigned task.

## 2. Netzaufbau mit PTV Vissim



- **Strecken** können im **Strecken-Modus** durch **Rechtsklick** und **„Drag-and-Drop“** erzeugt werden. Das **Strecken-Fenster** öffnet sich beim Erzeugen einer Strecke oder später durch **Doppel-Linksklick** auf eine Strecke.



- Die **Geometrie** kann im **Strecken-Modus** durch **„Drag-and-Drop“** der Polygonpunkte geändert werden. Durch **Rechtsklick** auf eine ausgewählte Strecke können neue Polygonpunkte erzeugt werden.
- **Fahrstreifen** können nach dem Erzeugen einer Strecke oder später im **Strecken-Fenster** geändert werden. Im Strecken-Fenster können die **Anzahl Fahrstreifen** und deren Attribute geändert werden.
  - **Breite**



- **Verbinder** sind besondere Strecken und können im **Strecken-Modus** durch **Rechtsklick** auf eine bereits existierende und **„Drag-and-Drop“** zu einer weiteren Strecke erzeugt werden. Das **Verbinder-Fenster** öffnet sich beim Erzeugen eines Verbinders oder durch **Doppel-Linksklick** auf einen existierenden Verbinder.
- **Einordnen-Distanz** kann beim Erzeugen des Verbinders oder später im **Verbinder-Fenster** geändert werden.

**Figure A.7:** The introduction to PTV VISSIM used in the study given in German. It contains all infrastructure objects required to finish the assigned task.

---

### 3. Zuflüsse und Routen in PTV Vissim



- **Fahrzeug-Zusammensetzungen** können in der **Listen-Ansicht** durch klicken auf ‚Hinzufügen‘ erzeugt werden. Die **Listen-Ansicht** ist im Menü unter ‚Verkehr > Fahrzeug-Zusammensetzungen‘ zu finden.

- **V-Wunsch-Verteilung** kann in der **Listen-Ansicht** geändert werden.
- **Fahrzeugtyp** kann in der **Listen-Ansicht** geändert werden.



- **Fahrzeugzuflüsse** können im **Fahrzeugzuflüsse-Modus** durch **Rechtsklick** auf eine Strecke erzeugt werden. Nach dem Erzeugen öffnet sich die **Listen-Ansicht**. Alternativ ist die **Listen-Ansicht** im Menü unter ‚Listen > Individualverkehr > Zuflüsse‘ oder durch **Doppel-Linksklick** auf einen Fahrzeugzufluss zu finden.

- **Belastung** kann in der **Listen-Ansicht** geändert werden.
- **Fahrzeug-Zusammensetzung** kann in der **Listen-Ansicht** geändert werden.










- **Routenentscheidungen (statisch)** können im **Fahrzeugrouten-Modus** durch **Rechtsklick** auf eine Strecke erzeugt werden. Nach dem Erzeugen der Entscheidungspunkt erzeugt wurde können beliebig viele **Routenziele** durch erneutes **rechtsklicken** auf die gewünschten Strecken erzeugt werden. Wenn alle gewünschten Routen erzeugt wurden, kann der Vorgang durch **klicken neben einer Strecke** oder ‚Esc‘ beendet werden.

- **Position** der Routenentscheidung kann durch ‚Drag-and-Drop‘ oder in der **Listen-Ansicht** geändert werden. Die **Listen-Ansicht** ist im Menü unter ‚Listen > Individualverkehr > Routen > Statische Routenentscheidungen‘ oder durch **Doppel-Linksklick** auf die Routenentscheidung zu finden.
- **Routen** können in der **Listen-Ansicht** geändert werden. Die **Listen-Ansicht** ist im Menü unter ‚Listen > Individualverkehr > Routen > Statische Routen‘ oder durch **Doppel-Linksklick** auf das Routen-Ziel zu finden.
  - **Zielposition** kann durch ‚Drag-and-Drop‘ oder in der **Listen-Ansicht** geändert werden.
  - **Relative Belastung** kann in der **Listen-Ansicht** geändert werden.

**Figure A.8:** The introduction to PTV VISSIM used in the study given in German. It contains all demand objects required to finish the assigned task.

#### 4. Auswertung mit PTV Vissim

- 
  - **Reisezeitmessungen** können im **Fahrzeugreisezeiten-Modus** durch **rechtsklicken** auf die zwei gewünschten Strecken erzeugt werden. Das **Reisezeitmessung-Fenster** öffnet sich beim Erzeugen einer Reisezeitmessung oder durch **Doppel-Linksklick** auf eine existierende Reisezeitmessung.
    - **Start-/End-Position** kann beim Erzeugen der Reisezeitmessung oder später durch **„Drag-and-Drop“** oder im **Reisezeitmessung-Fenster** geändert werden.
  - 
    - **Fahrzeugreisezeit-Ergebnisse** sind im Menü unter **„Auswertung > Ergebnislisten > Fahrzeugreisezeit-Ergebnisse“** zu finden.
      - **„Nach Simulationslauf in Datei Schreiben“** kann durch klicken auf die Schaltfläche aktiviert werden.
- 
  - **Verlustzeitmessungen** können in der **Listen-Ansicht** durch klicken auf **„Hinzufügen“** erzeugt werden. Die Listen-Ansicht ist im **Menü** unter **„Auswertung > Messungs-Definition > Verlustzeitmessungen“** zu finden.
    - **Fahrzeugreisezeitmessungen** können in der **Listen-Ansicht** geändert werden.
  - 
    - **Verlustzeitmessungen-Ergebnisse** sind im Menü unter **„Auswertung > Ergebnislisten > Verlustzeitmessungen-Ergebnisse“** zu finden.
      - **„Nach Simulationslauf in Datei Schreiben“** kann durch klicken auf die Schaltfläche aktiviert werden.
- 
  - **Messquerschnitte** können im **Messquerschnitte-Modus** durch **Rechtsklick** auf eine Strecke erzeugt werden. Die **Liste-Ansicht** ist im Menü unter **„Listen > Messungen > Messquerschnitte“** oder durch **Doppel-Linksklick** auf einen Messquerschnitt zu finden.
    - **Position** kann durch **„Drag-and-Drop“** oder in der **Listen-Ansicht** geändert werden.
- 
  - **Querschnittsmessungen** können in der **Listen-Ansicht** durch klicken auf **„Hinzufügen“** erzeugt werden. Die Listen-Ansicht ist im **Menü** unter **„Auswertung > Messungs-Definition > Querschnittsmessungen“** zu finden.
    - **Messquerschnitte** können in der **Listen-Ansicht** geändert werden.
  - 
    - **Querschnittsmessungs-Ergebnisse** sind im Menü unter **„Auswertung > Ergebnislisten > Querschnittsmessungs-Ergebnisse“** zu finden.
      - **„Nach Simulationslauf in Datei Schreiben“** kann durch klicken auf die Schaltfläche aktiviert werden.

**Figure A.9:** The introduction to PTV VISSIM used in the study given in German. It contains all evaluation objects required to finish the assigned task.

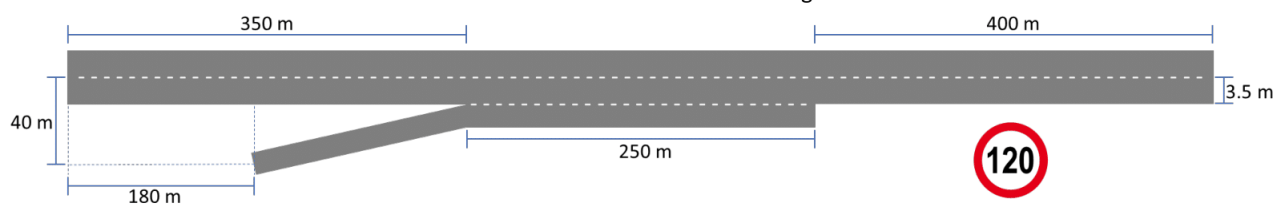


- 
- **Auswertungskonfiguration** kann im **Konfigurations-Fenster** geändert werden. Das Konfigurations-Fenster ist im **Menü** unter ‚**Auswertung > Konfiguration**‘ zu finden.
    - **Ergebnisattribute-Tab**
      - **Querschnitts-, Fahrzeugreisezeit-, Verlustzeitmessungen** können mit den **Auswahlboxen** aktiviert werden.
      - Das **Intervall** kann jeweils geändert werden.

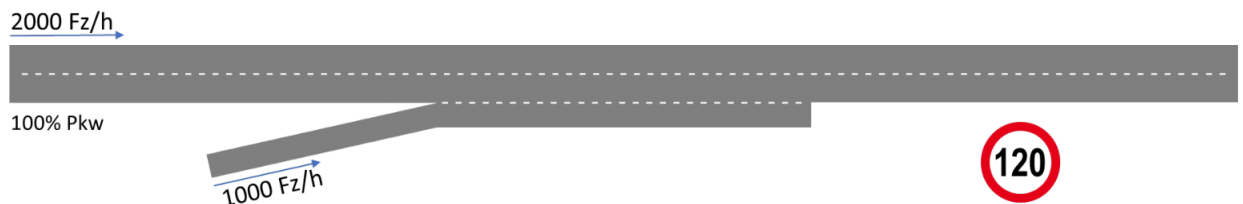
**Figure A.10:** The introduction to PTV VISSIM used in the study given in German. It describes the evaluation configuration which is required to finish the assigned task.

## Studie zur konsistenten Modellierung mit PTV Vissim und SUMO – Autobahnauffahrt

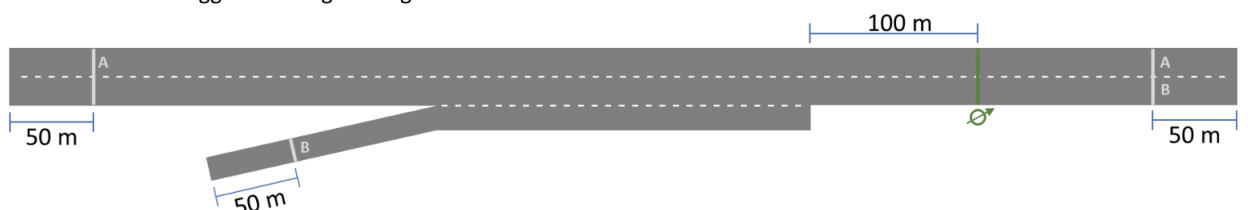
In dieser Aufgabe soll eine **Autobahnauffahrt** modelliert werden. Dazu soll ein **1 km** langer, **gerader** Ausschnitt einer Autobahn mit **2 Fahrstreifen** modelliert werden. Nach **350 m** der modellierten Strecke beginnt der **250 m** lange Beschleunigungsstreifen. Alle Fahrstreifen und der Beschleunigungsstreifen haben eine **Breite** von **3.5 m**. Auf der modellierten Strecke gilt eine **Geschwindigkeitsbegrenzung** von **120 km/h (33.33 m/s)**. Die Rampe, auf der Fahrzeuge auf die Autobahn auffahren können, soll als **gerade** Strecke mit **einem Fahrstreifen** modelliert werden. Die Rampe soll **180 m** nach Beginn der Autobahnstrecke und mit **40 m** Abstand von der Fahrbahnmitte beginnen.



Der Zufluss auf der Autobahn beträgt **2000 Fz/h**. Der Zufluss auf der Rampe beträgt **1000 Fz/h**. Dabei soll angenommen werden, dass es sich bei allen Fahrzeugen um **PKW** handelt.



Zur Auswertung der modellierten Autobahnauffahrt, soll die **Reisezeit** und die **Verlustzeit** der Fahrzeuge bestimmt werden. Diese sollen für auffahrende Fahrzeuge (B) und Fahrzeuge im Hauptstrom (A) getrennt gemessen werden. Die gemessenen Zeiten sollen in **5 min Intervallen** aggregiert werden. Der Bereich, in dem die Reisezeit gemessen wird, soll jeweils **50 m** nach dem Anfang der modellierten Strecke beginnen und **50 m** vor dem Ende der Strecke aufhören. Außerdem soll an einem Messquerschnitt **100 m** nach dem Ende des Beschleunigungsstreifens die Fahrzeuggeschwindigkeiten gemessen werden.



**Figure A.11:** The modeling task used in the study given in German.

Um die Anwendung zu bewerten, füllen Sie bitte den nachfolgenden Fragebogen aus. Er besteht aus Gegensatzpaaren von Eigenschaften, die die Anwendung haben kann. Abstufungen zwischen den Gegensätzen sind durch Kreise dargestellt. Durch Ankreuzen eines dieser Kreise können Sie Ihre Zustimmung zu einem Begriff äußern. Entscheiden Sie möglichst spontan. Es ist wichtig, dass Sie nicht lange über die Begriffe nachdenken, damit Ihre unmittelbare Einschätzung zum Tragen kommt.

Bitte geben Sie nun Ihre Einschätzung ab. Kreuzen Sie bitte nur einen Kreis pro Zeile an.

	1	2	3	4	5	6	7		
unerfreulich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	erfreulich	1
unverständlich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	verständlich	2
kreativ	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	phantasielos	3
leicht zu lernen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	schwer zu lernen	4
wertvoll	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	minderwertig	5
langweilig	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	spannend	6
uninteressant	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	interessant	7
unberechenbar	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	voraussagbar	8
schnell	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	langsam	9
originell	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	konventionell	10
behindernd	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unterstützend	11
gut	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	schlecht	12
kompliziert	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	einfach	13
abstoßend	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	anziehend	14
herkömmlich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	neuartig	15
unangenehm	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	angenehm	16
sicher	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unsicher	17
aktivierend	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	einschläfernd	18
erwartungskonform	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	nicht erwartungskonform	19
ineffizient	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	effizient	20
übersichtlich	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	verwirrend	21
unpragmatisch	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	pragmatisch	22
aufgeräumt	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	überladen	23
attraktiv	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unattraktiv	24
sympathisch	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unsympathisch	25
konservativ	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	innovativ	26

**Figure A.12:** The User Experience Questionnaire [51] used in the study.

Um die Anwendung zu bewerten, füllen Sie bitte den nachfolgenden Fragebogen aus. Er besteht aus einfachen Aussagen zur Anwendung. Durch Ankreuzen eines der Kreise können Sie Ihren Grad an Zustimmung zu der jeweiligen Aussage äußern.

Bitte geben Sie nun Ihre Einschätzung des Verfahrens ab. Kreuzen Sie bitte nur einen Kreis pro Zeile an.

Aussage	Stimme gar nicht zu			Stimme voll zu		
	1	2	3	4	5	
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	1
Ich empfinde das System als unnötig komplex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	2
Ich empfinde das System als einfach zu nutzen.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	3
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	4
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	5
Ich finde, dass es im System zu viele Inkonsistenzen gibt.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	6
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	7
Ich empfinde die Bedienung als sehr umständlich.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	8
Ich habe mich bei der Nutzung des Systems sehr souverän gefühlt.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	9
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	10

Um die Anwendung zu bewerten, beantworten Sie, falls möglich, die folgenden Fragen.

1. Nennen Sie drei bis fünf Punkte, die Ihnen an dieser Anwendung am besten gefallen haben:

---



---



---



---



---

2. Nennen Sie drei bis fünf Punkte, die Ihnen an dieser Anwendung am wenigsten gefallen haben:

---



---



---



---



---

**Figure A.13:** The System Usability Scale [52] used in the study.