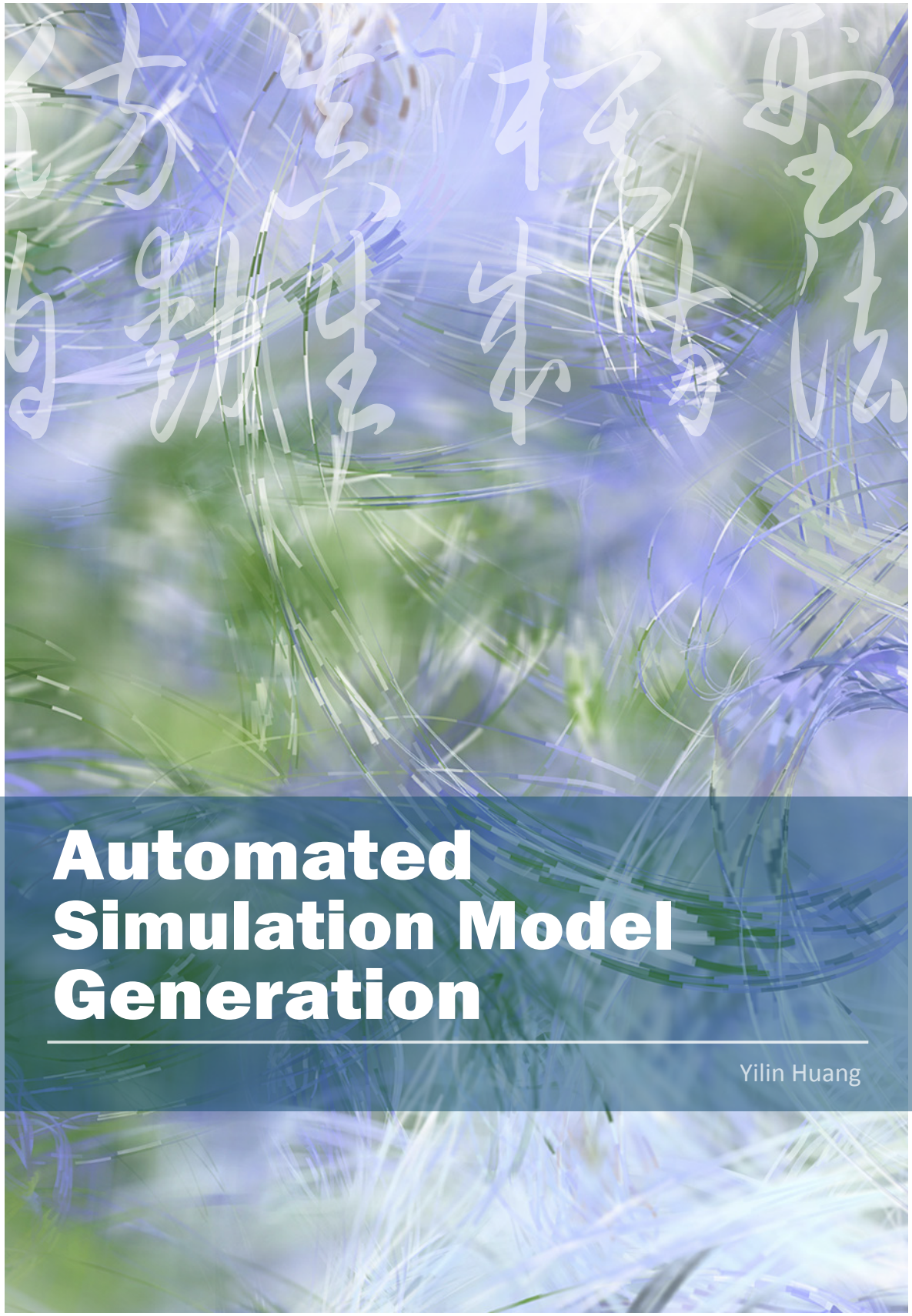


Automated Simulation Model Generation

Yilin Huang

Automated Simulation Model Generation

Automated Simulation Model Generation

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.Ch.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op dinsdag 5 november 2013 om 12.30 uur
door

Yilin HUANG
Diplom-Ingenieurin (FH)
geboren te Shanghai, China

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. A. Verbraeck

Copromotor: Dr. M.D. Seck

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. A. Verbraeck	Technische Universiteit Delft, promotor
Dr. M.D. Seck,	Technische Universiteit Delft, copromotor
Prof. dr. F.M.T. Brazier	Technische Universiteit Delft
Prof. dr. ir. P.M. Herder	Technische Universiteit Delft
Prof. dr. A. Tolk	Old Dominion University
Prof. dr. H. Vangheluwe	Universiteit Antwerp
Prof. dr. S. Straßburger	Technische Universität Ilmenau
Prof. dr. ir. W.A.H. Thissen	Technische Universiteit Delft, reservelid

Dissertation
Automated Simulation Model Generation

Yilin Huang
ME@YILIN.INFO

Section Systems Engineering
Faculty of Technology, Policy and Management
Delft University of Technology
The Netherlands

Typeset with LaTeX 2ε
Cover image by Patrick Gunderson www.theorigin.net
Cover design by Yilin Huang
Printed by CPI Wöhrman Print Service www.wps.nl

ISBN/EAN 978-94-6203-461-7

To my family

謝給我的家人

Acknowledgment

I have been extremely fortunate in the past years to know a number of interesting people with whom I spent quality time both at work and in my private life. I would like to take this opportunity to express my sincere gratitude for their time, professional and personal help and support, company and friendship.

I held lengthy and thorough discussions with my promoter Alexander Verbraeck and my supervisor Mamadou Seck. Alexander offered me a position at Delft allowing me to reunite with my then-boyfriend-now-husband while pursuing my PhD. As a promoter, he is sharp-minded, critical and encouraging. He gave me guidance and at the same time enough freedom to conduct my research. Mamadou, who arrived at Delft the same year as I did, is gentle and philosophical. Besides work, we also had interesting discussions about life and religion. From both Alexander and Mamadou, I could not have wished for more challenging ideas, constructive criticisms and support.

At Delft, interactions with colleagues were pleasant. Diones Supriana, Everdine de Vreede-Volkers and Sabrina Ramos Rodrigues were at all times helpful with administrative issues. As a fresh PhD, I had much fun with earlier PhD fellows and roommates, Rafael Gonzáles, Michele Fumarola and Jan-Paul van Staalduinen, who are smart gentlemen. Special thanks to Michele for his patience and understanding with my indecisiveness on a certain issue. PhDs and PostDocs who joined the section later are all friendly chaps who were good company forming an international and vibrant group, Deniz Çentinkaya, Tanja Buttler, Jordan Janeiro, Çağri Tekinay, Kassidy Clark, Evangelos Pournaras, Maartje van den Bogaard, Stefan Knoll, Shalini Kurapati, Farideh Heidari, Yakup Koç, Mingxin Zhang. Thanks for the Ouzo, Raki, Caipirinha, Sake, Whiskey, coffee, tee, chocolates, BBQ, Chinese food, parties and other cheerful gatherings. I enjoyed lunch breaks, sweet treats and discussions with colleagues. Martijn Warnier's inquiries about my dissertation progress, Michel Oey's concert flyers, Jos Vrancken's birthday quiz, Sander van Splunter's cookies ... I will remember all.

HTM provided me with invaluable resources for the research project. From data and documents to productive meetings, many people offered generous help. Hilbert Veldhoen had his important role in the project. We had uncountable numbers of discussions, exchanging ideas and experiences. He organized visits for me to different departments and introduced me to many experts with whom I collaborated. Tom van der Heijden was always very responsive to my questions and gave me good hints to solve problems concerning the EBS database. The discussions with Niels van Oort and Peter

Scheepmaker were delightful and full of information. The latter kindly invited me to the tram museum he managed. The others I would like to thank include Peter Tros, Marc Drost, Anne-Wil Boterman, the members in the validation panel and the HTM board, and those that I encountered in meetings and on other occasions whose names I failed to note down (for which I apologize).

My old friends in Shanghai, the Viennese family Lotte, Peter and Alexander Smrha, who cared and are still caring for me as if I was their own, friends from Erlachplatz, FH and TU in Vienna, those I encountered in The Hague, Delft, and more recently at the climbing hall, all have brought many joyful moments in my life. My family is always unconditionally loving and supportive. I thank my husband for being who he is and particularly for his understanding and trust. I do not feel I can adequately express my gratefulness to my family and friends in a few words. But please keep in mind that wherever my life path leads to, I will stand by you whenever you are in need.

Yilin Huang, Delft, July 2013

Contents

1

Towards Automated Simulation Model Generation

SIMULATION is a useful method for analyzing the design and operation of complex systems (SHANNON 1975, SOL 1982, LAW 2007). It can be used to imitate the operation of a real system by executing a model of that system over time (BANKS et al. 2010), nowadays often with the use of computers. Simulation models of this kind are dynamic, as opposed to static models which do not simulate systems change over time trajectories. In this research, we are interested in automated generation of dynamic computer-based simulation models.

1.1 Research Context

One of today's challenges in the field of *Modeling and Simulation* (M&S)[1] is the need to model and simulate increasingly larger and more complex systems (CROSBIE 2010). It currently takes too long to develop and experiment with models, not to mention the

[1] M&S is also known as *simulation modeling.*

high cost and human resource involved (FOWLER and ROSE 2004, BANKS et al. 2010). Many examples can be found in production and manufacturing (FOWLER and ROSE 2004), supply chains (LONGO 2011), air transportation (WIELAND and PRITCHETT 2007), health care (MIELCZAREK and UZIALKO-MYDLIKOWSKA 2012), to name just a few.

1.1.1 Motivation

Modeling refers to the full scope of activities in model development. It is central to the efficacy of simulation (PAGE 1995). There is a rich history of efforts to improve the effectiveness and efficiency of the modeling process in order to better the overall M&S process, e.g., developing simulation languages and user interfaces for modeling, and developing domain specific simulators (FOWLER and ROSE 2004). While all these developments have significantly reduced the time and effort to build models, there is still considerable room for improvement (ibid.).

One opportunity to reduce time and effort in modeling is to use the available data of a system to automatically or semi-automatically generate simulation models (FOWLER and ROSE 2004, BERGMANN and STRASSBURGER 2010). Regardless of whether modeling is performed by humans or by automation, data sources include data acquired by observation and measurement, as well as documents about a system (SHANNON 1975). The former type of data can be used, e.g., to determine the appropriate input distribution and to validate simulation output data; the latter type can be used, e.g., to define and configure simulation models.

Increased Data Availability The availability of both types of data has increased along with the advances in sensor technology as well as the more popular use of computer-aided technologies such as CAD, CAE, ERP and MES systems[2] (GLOTZER et al. 2010). The increased availability of data has rendered automation more attractive. On the one hand, the increased availability of data allows for a higher degree of automation in modeling as more useful information becomes accessible in digital forms. On the other hand, the increased amount of data often requires automation because the data can no longer be handled manually in an effective and efficient manner (ibid.).

Automation in Modeling In this thesis, we study *Automated Model Generation* (AMG). The goal is to develop a method that can automatically generate simulation models. AMG is a relatively new research field with early works dating around 1990s. Some literature calls it *automated modeling* (AMSTERDAM 1993, NAYAK 1995, XIA and SMITH 1996, GRANDA and MONTGOMERY 2003).

Many works in AMG use circuit design schematics (WASYNCZUK and SUDHOFF 1996, EECKELAERT et al. 2004, LITTLE et al. 2010), SysML (CAO et al. 2012, JOHNSON et al. 2012), or bond graphs (GRANDA and MONTGOMERY 2003, UMESH RAI and UMANAND 2009, ROYCHOUDHURY et al. 2011, TIAN et al. 2012, ZUPANCIC and SODJA 2012) to generate

[2] Computer Aided Design (CAD); Computer Aided Engineering (CAE); Enterprise Requirements Planning (ERP); Manufacturing Execution Systems (MES).

simulation models for physical systems such as circuits, hydraulics and mechatronics, or for biochemical processes and manufacturing (FERNEY 2000, THOMASETH 2003, MUELLER 2007, ROMAN and SELISTEANU 2012). The simulation models are generated based on model-like descriptions which already contain the required structure information. GELSEY (1990, 1995) and LEVY et al. (1997) apply advanced reasoning methods to determine model structures (i.e., the interrelations of components) based on pre-specified component descriptions. In these works, the input for AMG, i.e., the systems specification, is prepared for the AMG, by which the pre-specified parts or structures match the model parts and structures. It is an ideal situation but not always applicable to model large and complex systems.

There are some recent works that use existing data for AMG. An often used approach is to define general/generic models or model templates so that specific model instances can be created through parameter configuration. For example, BRAUSE (2004) can select differential equation models of minimum description length by parameter pruning (i.e., unnecessary parameters become zero). HARRISON et al. (2004) and LUCKO et al. (2010) use data to configure parameters such as the amount of resources and time in workflow process models in Arena® and WorkSim®. In WANG et al. (2011), automobile general assembly plant models can be generated based on the physical layout data and the production data of the plant; but still, the model selection is parameter-based.

Research Opportunity To our knowledge, there has not been works of AMG that can generate simulation models with flexible structures using existing data. By "generating simulation models with flexible structures", we mean that simulation models are not generated by parameter-based configuration on a pre-specified model structure, but a model with a (new) structure is dynamically constructed according to the existing data. To demonstrate why this is needed, we first give an example.

1.1.2 An Example: HTM Cases

HTM (*Haagsche Tramweg Maatschappij*, www.htm.net) is a public transport operator based in The Hague, the Netherlands. The organization often uses M&S to study the design and operation of their light-rail network[3] (VELDHOEN 2009), e.g.,

- the infrastructure and control at intersections (KANACILO and VERBRAECK 2006, 2007);
- the design of new infrastructure and operation (KAMERLING 2007, HUANG et al. 2010);
- the depot capacity and the vehicle planning on the deadhead-kilometer (non-value added trips) (CAI 2011);
- the strategies in the design of infrastructure networks, service networks and timetables (VAN OORT 2011).

[3] The network in The Hague covers over 150 km^2 with fourteen scheduled tram lines, 140 km tracks and 540 stops.

Every year, a number of M&S studies is initiated by the organization. New simulation models are needed to study new or different parts of the light-rail network and operation. The models are different in the sense that they have different infrastructure layouts, services and timetables, etc., yet they share similar underlying concepts. Developing the simulation models is labor intensive and time consuming – three to six months for small projects and over a year for large ones.

The organization possesses a large amount of data, from infrastructure design, service plans and timetables to the (sensor collected) passenger counts and GPS data. Using the data in modeling could provide useful information about the system and help improve the validity of models. However, the more data modelers use, the longer it takes to develop the models. Constrained by time and cost, a large part of the available data is unused.

There is a huge interest from the organization to improve the situation. Because the underlying study goals of the simulation projects in the organization are often very similar, i.e., to study the infrastructure, control strategies and timetables in relation with, e.g., the quality, reliability and robustness of the services, it is desirable to develop an automated routine that can generate the simulation models with different structures and to reuse some previous modeling solutions. The availability of data in the organization makes this kind of automation possible.

1.1.3 Problem Statement

Indeed, many organizations are facing a similar situation. On the one hand, they often need simulation models which take a long time to develop and incur high costs (WIELAND and PRITCHETT 2007, LONGO 2011). On the other hand, more and more data has become available in organizations which could provide useful information for modeling, however, much of it is unused (GLOTZER et al. 2010). To benefit from the data and improve the situation, there is a need for a method that can automatically use the existing data to generate simulation models.

Using existing data for AMG is very different from using data that is specifically prepared for the purpose of model generation. In the latter case, the data already contains the right content and structure of information required for model generation (e.g., FERNEY 2000, GRANDA and MONTGOMERY 2003, MUELLER 2007, ROMAN and SELISTEANU 2012). In existing data, the information that can be directly used for model generation is often not readily available (COBP 2002). The data may need to be transformed (or rewritten) in content and structure. The transformation should eventually lead to a model structure according to which a simulation model can be generated.

Model structure variations can be achieved with parameter-based approaches (e.g., BRAUSE 2004, WANG et al. 2011). In such cases, all possible model structures have to be pre-specified, which is not convenient and can be impractical when the models are complex (with many parts or components). The ability to dynamically construct structure variations provides more flexibility. Hence, we choose to generate model structures dynamically during the AMG.

How to use formal methods to achieve the above goals is the aim of this research. To summarize, the research presented in this thesis differs from previous works of AMG in at least two aspects:

- We aim at using existing data where the model structures are not directly defined.
- We aim at generating simulation models with flexible structures which are not pre-specified before model generation.

1.2 Research Objective and Questions

The discussion so far depicted the motivation and problem situation of this research. The research is motivated by the fact that modeling large and complex systems takes a long time and is costly. As more data becomes available in many organizations, the issue of how to efficiently and effectively use the data to help modeling has attracted more attention. AMG is studied by previous works in different ways (§ 1.1.1). We position this research at a niche. We posit that there is both a need and a lack of works in tackling the problem of generating simulation models with flexible structures using existing data. Therefore, we set the research objective to provide a method to tackle this problem as following.

Research Objective *To provide a method that automatically generates simulation models with flexible structures using existing data assuming that these simulation models are intended for a certain domain.*

The objective states what is given to the method and what is to be expected from it. When we treat AMG as a function, then the existing data is the input of the function and the generated simulation model is the output. The research objective is to define an appropriate transformation function. To reach the research objective, we first need to know the nature of the input and the output. For example, what information is or should be carried by the existing data? What information is required for generating a simulation model? And how to define a flexible model structure to accommodate changes in the data within a given degree? These tasks can be phrased as the first two research questions.

Research Question 1 *What is a good way to define flexible structures for simulation models in order to achieve the research objective?*

Research Question 2 *What are the requirements for the data in order to achieve the research objective?*

With this knowledge, we can embark upon defining the transformation function. We first identify the requirements for the function, i.e., what functionalities it should provide, and then design a method that can deliver these functionalities[4].

[4] Non-functional requirements are as important; however, they are not the focus of this research.

Research Question 3 *What functionalities should a method provide in order to automatically generate simulation models with flexible structures using existing data?*

And finally, we evaluate the method by assessing the quality of the simulation model generated.

Research Question 4 *What is the quality of the simulation models generated by the method?*

We use cases in the domain of light-rail transport to study the AMG method.

1.3 Research Strategy

To be an effective systems scientist, we must at the same time be both a holist, looking at a system as a whole, and a reductionist, understanding a system in more detailed forms (M'PHERSON 1974, FLOOD and CARSON 1993).

1.3.1 Research Philosophy

The philosophical basis of research is important for numerous reasons. It can assist in understanding notable issues: the interrelation between ontological, epistemological, and methodological[5] levels of inquiry (PROCTOR 1998, CRESWELL 2003, TOLK 2013). It is significant in reference to research methodology: (1) to clarify the overall research strategy and to refine and specify the research methods; (2) to enable and assist the evaluation of different methods and avoid inappropriate use; and (3) to help researchers to be creative and innovative in either the selection or the adaptation of methods (EASTERBY-SMITH et al. 2001).

In literature, positivism and interpretivism are two common and often seen as opposing views of research philosophy. Table 1.1 lists their basic differences. Some other views include postpositivism, critical theory, constructivism, postmodernism, etc. What the critiques of different philosophical views have brought, as MINGERS (2006) puts it, is the refutation of any form of monism. MINGERS (ibid.) further states that "we have to recognize a plurality ... This should not be seen as some sort of slide into relativism, but rather a recognition of the amazing complexity of the world we are trying to understand." CROSSAN (2003) argues that sometimes the distinction between quantitative and qualitative philosophies and research methods is overstated, and in contemporary research they are indeed frequently used in conjunction. WEBER (2004) contends and exemplifies that the distinction between positivism and interpretivism is not clear-cut, and the rhetoric of positivism versus interpretivism no longer serves a useful purpose.

Combining different intervention and research methods can have many benefits in dealing with different dimensions of system problems. The choice of approach may be

[5] The ontological question: what is the form and nature of reality and, therefore, what can be known about it? The epistemological question: what is the nature of the relationship between the knower or would-be knower and what can be known? The methodological question: how can the inquirer (would-be knower) go about finding out whatever he or she believes can be known? GUBA and LINCOLN (1994)

Meta-theoretical Assumptions	Positivism	Interpretivism
Ontology	Person (researcher) and reality are separate.	Person (researcher) and reality are inseparable (life-world).
Epistemology	Objective reality exists beyond the human mind (objective, tangible, single).	Knowledge of the world is intentionally constituted through a person's lived experience (socially constructed, multiple).
Methodology	Hypothetical-deductive approach (experimental design)	Holistic-inductive approach (naturalistic inquiry)
Research Object	Research object has inherent qualities that exist independently of the researcher.	Research object is interpreted in light of the structure of meaning of a researcher's lived experience.
Goal of Research	Explanation, strong prediction	Understanding, weak prediction
Focus of Interest	What is general, average and representative	What is specific, unique and deviant
Research Methods	Lab and field experiments, survey, case study, simulation, etc. (seek empirical data and analyze statistically).	Hermeneutics, phenomenology, ethnography, case study, action research, etc.
Subject-Researcher	Rigid separation	Interactive, cooperative
Knowledge Generated	Laws: absolute (time, context, and value-free)	Meanings: relative (time, context, culture, and value-bound)
Desired Information	How many people think and do a specific thing, or have a specific problem?	What do some people think and do, what kind of problems are they confronted with and how do they deal with it?
Theory of Truth	Correspondence theory of truth: one-to-one mapping between research statements and reality.	Truth as intentional fulfillment: interpretations of research object match the lived experience of the object.
Validity	Certainty: data truly measures reality.	Defensible knowledge claims.
Reliability	Replicability: research results can be reproduced.	Interpretive awareness: researchers recognize and address implications of their subjectivity.

Table 1.1: Positivism and interpretivism – basic differences (WEBER 2004, DECROP 2006)

dependent on the nature of the phenomena of interest, the goal of the research, the level and nature of the questions, researchers' experience and personal beliefs, and practical considerations related to the research environment and the efficient use of resources (CROSSAN 2003).

This research aims at AMG with a focus on flexible model structure. Naturally posit-

ivism is the main philosophical view to follow during the method and tool development. However, systems often involves human actors, and domain experts have their perceptions, values and interests, which must not be neglected. A deterministic perspective may hinder systems understanding and human interaction. Therefore, a positivist's view combined with interpretive elements is appropriate to gather information of systems knowledge and to evaluate the research outcome as to avoid over-quantification.

1.3.2 Research Approach

M&S and *systems science* are two fields that are intimately entwined (WYMORE 1967, KLIR and ELIAS 2003, ZEIGLER 2003). *Systems engineering* is the engineering subset of systems science, which deals with the design of systems in a broad sense (SAGE 1992, BLANCHARD and FABRYCKY 2011). In systems engineering, there exists a number of development processes, but most are grounded in one of the following three seminal processes (ESTEFAN 2008):

(1) ROYCE (1970)'s waterfall model which elaborates an iterative and incremental relationship between successive development phases,

(2) BOEHM (1988)'s spiral model which has various refinements of the waterfall model and emphasizes on cyclic development, and

(3) FORSBERG and MOOZ (1992)'s "Vee" model which emphasizes on systems decomposition and alternative design concepts.

In our view, the underlining concepts of these processes are not inconsistent. They can be used in a combined manner, since the characteristics of the processes stated above are all important for the development of solutions for the research problems.

An M&S system can be deemed as an information system (JACOBS 2005). In information systems research, two foundational and complementary paradigms are *design science* and *behavioral science* (MARCH and SMITH 1995, HEVNER et al. 2004). The former has its roots in engineering and the sciences of the artificial (SIMON 1996), seeking to create "what is effective" for problem solving (HEVNER et al. 2004). The latter has its roots in natural sciences research methods, seeking to know "what is true" for explanation and/or prediction (ibid.). The two paradigms are combined by HEVNER et al. (ibid.) who advocate for an iterative and incremental design process solving real-life problems using grounded scientific theory.

We therefore approached the research problems from a combined perspective of systems engineering and design science, with a focus on M&S, through which an AMG method (an artifact) should be designed to generate simulation models (artifacts) that are used to support explanation and/or prediction of systems of a certain domain.

1.3.3 Research Instruments

For systems understanding (which refers to the domain knowledge), we used *literature review* and *semi-structured interviews*. We studied existing literature and (HTM in-

ternal) documents relevant to the rail, light-rail and transport domain. The literature helps to substantiate the problems and suggests possible questions or hypotheses that need to be addressed (CRESWELL 2003). We also interviewed domain experts to gain first-hand domain knowledge to conduct the research. The questions and discussions mainly focused on modeling related aspects such as systems behavior and structure. The experts were also allowed to talk openly about any topics which they deemed potentially important for the research without the use of specific questions (ibid.).

For solution design and development, we used *literature review*, *data analysis*, *semi-structured interview*, *experiments* and *case studies*. Literature review was mainly conducted in the domains of systems theory, M&S, information systems, graph transformation and related works. Besides what is mentioned above, the literature was also used to present results of similar studies and to relate this research to the ongoing dialogue in the literature (ibid.). A theoretical approach was used for our solution whenever possible.

Data analysis is an ongoing process during research (ibid.). We performed data analysis (1) on the data used as the input for the AMG to identify data quality issues and possible solutions, (2) on the data of the model output to verify model behaviors, and (3) on the data from the real system to validate model behaviors. Some data analysis sessions were performed together with the domain experts.

As in gathering information about the relevant domain knowledge, we used semi-structured interviews to discuss and evaluate the design concepts with domain experts. Experiments and case studies were conducted with the simulation model to evaluate the design. In software engineering, experiments are typically used to explore relationship among data to evaluate the accuracy of methods or to validate measures; and case studies are important for the industrial evaluation of the methods and tools (SJOBERG et al. 2007).

For the final model validation, we used questionnaires and panel discussions with domain experts who evaluated the structure validity of the domain simulation models.

1.4 Thesis Outline

In § 2, we present a literature-based and systems-theory-rooted foundation of M&S upon which the later chapters are developed. The main concepts include levels of systems knowledge and specification, meta-models, component-based models and modeling formalisms.

In § 3, we provide an outlook on the AMG method. We first discuss the research questions and propose a set of constructs for the AMG method based on § 2. We then give a modeling example to elucidate data quality issues for an AMG method, and discuss a number of data quality categories and criteria. Finally, we propose three steps for an AMG process.

The AMG method proposed by this research is presented throughout § 4, § 5 and § 6. We use cases in the domain of light-rail transport to study the AMG method. In § 4, we discuss the theory, design concepts, and research findings related to develop-

ing a simulation library that is fit for use for AMG. A domain model component library developed for the domain of light-rail transport is presented. In § 5, we discuss graph transformation theory, and present step-by-step the transformation rules for the AMG method. To complement the AMG method, in § 6, we present a model calibration procedure, and discuss a calibration test case.

The simulation models generated by the AMG method developed in this research was validated by an expert panel. The results are discussed in § 7.

In § 8, we summarize the thesis, high-light the research findings, show the practical relevance of this research and comment on future research.

2

Foundations of Systems and Simulation Modeling

THE STRONG TIES between *systems theory* (VON BERTALANFFY 1950, LASZLO and KRIPPNER 1998) and M&S have been elaborated by many authors (WYMORE 1967, ZEIGLER 2003, ÖREN and ZEIGLER 2012, TOLK et al. 2013b). The two are intimately entwined — success in the first became bound up with another kind of success in the second (ZEIGLER 2003). After all, a simulation model is a formal representation of relevant systems knowledge used to study a system.

In this chapter, we present a literature-based and systems-theory-rooted foundation of M&S upon which the later chapters are developed. The chapter begins with grounding concepts and theories of systems and models in § 2.1, among which the organization and specification of systems knowledge at distinct epistemological levels are explained.

We proceed with discussion about meta-models and component-based models in § 2.2 and § 2.3 correspondingly. The former is relevant to this thesis as meta-models allow for abstract representations of model structures of a class of models instead of just one model. The latter is a promising modeling concept, not only because it promotes

the development of ready-to-use building blocks for concrete model instantiations, but also because it is inline with the systems theoretical viewpoint of dealing with complexity, among many other benefits.

The final section § 2.4 of this chapter brings in another important issue in M&S, that is: modeling formalisms. We briefly discuss some formalism classes and formalisms in literature, and give the rationale for choosing the *Discrete Event System Specification* (DEVS) as the underlying modeling formalism for our model specifications. The DEVS formalism and its principles are then presented.

2

2.1 Systems and Models

2.1.1 What are They?

In M&S, a model represents "a portion of the real world" (SHANNON 1975), existing or not, under modeling interest. According to systems theorists such as ACKOFF (1973), FORRESTER (1976) and KLIR (2001), a **system** is a set of interrelated parts (or elements) of any kind that operate together for a common purpose. A **model** is an abstraction of a system intended to replicate some properties of that system (OVERSTREET 1982, PAGE 1995). This means that a model needs to possess three features (STACHOWIAK 1973, KÜHNE 2006):

Mapping feature A model is based on an *original system*, existing or non-existing. We may call the original system a *source system* or a *referent*.

Reduction feature A model only reflects a *relevant selection* of an original system's properties.

Pragmatic feature A model needs to be usable in place of an original system with respect to some *purpose*.

The relation between systems and models is seemingly straightforward. Some may hold a misconception of systems which would need some clarification. "A portion of the real world" (let us call it a referent) that is *perceived by an observer* as "a system" is in fact *an image* of the referent. This image is our *mental model*[1] of the referent, which already has reduced and somewhat organized complexity (SHANNON 1975, CHECKLAND 1981, FLOOD 1990, KLIR 2001). As such, it is reasonable to state that, strictly speaking, a system in our eyes and minds is a model, *not* the referent itself; though a simulation model is as well a model but a different one.

2.1.1.1 A Formal Definition of Systems

Before furthering the discussion, we shall give a formal definition of systems. One of the most generic definitions is given by WYMORE (1967). Wymore's definition of a system is

[1] The term *mental model* is suggested by the Scottish experimental psychologist Kenneth Craik (ZANGWILL 1980) — the mind carries a convenient small scale model of external reality that it uses to anticipate events (CRAIK 1943).

expressed by a 7-tuple (VANGHELUWE 2008, ÖREN and ZEIGLER 2012)

$$S = (T, X, \Omega, Q, \delta, Y, \lambda) \tag{2.1}$$

where

$T \subset \mathbb{R}_0^+$	is the time base
X	is the input set
$\Omega = \{\omega : T \to X\}$	is the input segment set or function
Q	is the state set
$\delta : \Omega \times Q \to Q$	is the transition function
Y	is the output set
$\lambda : Q \to Y$	is the output function

and the transition function satisfies the *composition property*

$$\forall \omega_1, \omega_2 \in \Omega, q \in Q : \ \delta(\omega_1 \bullet \omega_2, q) = \delta(\omega_2, \delta(\omega_1, q)) \tag{2.2}$$

where $\omega_1 \bullet \omega_2$ is a *concatenation* of two input segments. In essence, the property says that the state set must retain enough information to allow the system to continue from where it is left by the first input so that it can arrive at the same state it would have, had there been no interruption in the middle (ZEIGLER 2003).

2.1.1.2 Some Related Terms and Concepts

How a system and its complexity are perceived is strongly influenced by human perspectives and perceptions (FLOOD 1990). Approaching systems complexity, we habitually divide (or decompose) the system into less complex parts (or sub-systems) and analyze the parts and their relations, which are more comprehensible. This *decomposition* approach is well established in the systems theory and systems thinking literature (SIMON 1962, 1996, ACKOFF 1978, CHECKLAND 1999, KLIR and ELIAS 2003).

SIMON (1962) defines a *complex system* roughly as one made up of a large number of parts that interact in a non simple way. A **part** is the representation of some phenomena of the real world by a noun or a noun phrase that informed observers agree exists, or could exist, or whose existence may be worth assuming in order to gain insight (FLOOD and CARSON 1993). Each of these parts can be deemed a system and be formally defined, e.g., in the form of Eq. 2.1. Any characteristic quality or property ascribed to a part is a **variable** of that part (ibid.). A **state** is the values recognized for a set of variables (KLIR and ELIAS 2003).

A **relation** (or interaction) can be said to exist between two parts if the behavior of one is influenced or controlled by the other (JONES 1982). A **structure** defines the way in which the parts of a system can be related to each other (FLOOD and CARSON 1993). While structure is the *inner constitution* of a system; **behavior**, on the contrary, is the *outer manifestation* of a system (ZEIGLER et al. 2000). This means, when we view a system or a sub-system as a black box, its behavior is the relationship it imposes between its input trajectories and output trajectories, i.e., $X \to Y$ in Eq. 2.1 (ibid.).

Systems can be represented in the form of levels in *hierarchical* structures and organizations; more specifically, a system can be decomposed into smaller sub-systems, and systems can be composed to form larger systems (SIMON 1962, 1996, FLOOD and CARSON 1993, ZEIGLER et al. 2000). This hierarchy allows us to systematically reduce the breath of analysis from system to sub-systems to sub-sub-systems, etc., which increases the *level of resolution* of analysis (FLOOD and CARSON 1993). An important part of any study drawing upon systems is to choose an appropriate level of resolution to focus our attention on (ibid.). In simulation studies, the levels of resolution (or model details) are choices resulting from the intended use of the models (ZEIGLER et al. 2000, LAW 2007).

A simulation model can be defined to resemble a system in parts and relations. If properly designed and specified, the model is in a *morphism* to its system counterpart (ZEIGLER et al. 2000, KLIR 2001). We can achieve this when we have sufficient[2] knowledge about a system, i.e., *systems knowledge*, in terms of its parts, relations, structures, among others.

2.1.2 Sources of Systems Knowledge

A simulation model is a formal representation[3] of relevant systems knowledge. When a system under modeling interest is unknown or partially unknown to a modeler, he or she needs to first acquire sufficient systems knowledge[4] for the modeling goal. Systems knowledge can be learned from several sources; the knowledge learned can be overlapping and complementary, as shown in Figure 2.1.

Formal Systems Knowledge Formal knowledge is *explicit knowledge* that is articulated or presented with a unique meaning in a preservable form. Examples of formal systems knowledge are theories and theorems; mathematical models and simulation models are also formal systems knowledge. They can be recorded in books, articles and other media. An often used method for acquiring formal knowledge is literature review.

Informal Systems Knowledge Much systems knowledge is *implicit knowledge* in the heads of people who are familiar with the system, e.g., domain experts and users of the system. The mental models of the system constructed by these individuals may, often do, possess "first-hand knowledge" (MACK 1990) that is not covered by the formal sources. Clearly, informal systems knowledge is only usable after *elicitation* and *formalization*. Common methods for elicitation are meetings, interviews, questionnaires, etc.

[2] Sufficient to the extent demanded by the intended use of the model.

[3] A formal representation refers to a representation that is specified following a modeling formalism, see, e.g., § 2.4.1.

[4] This acquisition is meant for the gathering of existing systems knowledge, not the scientific discovery type of knowledge acquisition.

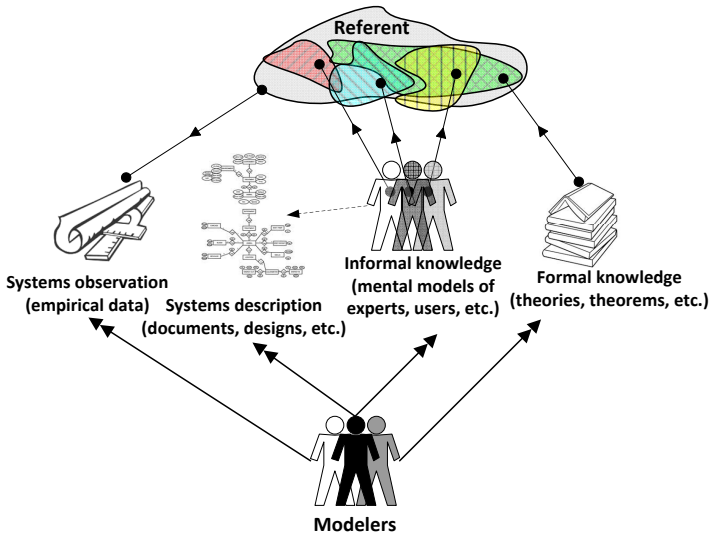


Figure 2.1: Sources of systems knowledge

The content of formal systems knowledge is readily accessible in its meaning and form. Its acquisition and exchange are often easier and faster than that of informal knowledge (WANG 2009). Nevertheless, a large portion of systems knowledge is learned informally. Hence, it is critical that modelers can identify and communicate effectively with people who possess systems knowledge and involve these people in the model development process (SHANNON 1975, BALCI 2012). Besides obtaining knowledge from the knowledgeable, we can also indirectly acquire systems knowledge through the analysis of different data. We classify different data into two categories: systems observation and systems description.

Systems Observation By systems observation we refer to the type of data that is obtained from recorded observation (or measurement) of a system. Some literature calls it *empirical data*. It potentially contains information about the behavior (i.e., the outer manifestation) of a system and its sub-systems. Following the definition of systems behavior previously stated, this type of data can describe the input and output, i.e., X and Y in Eq. 2.1, of a system and its sub-systems. Modelers often use it, e.g., to chose and fit input distribution and to validate output data of the models. With advances in data collection and storage technologies, many organizations have large amounts of systems observation.

Systems Description By systems description we refer to the type of data that potentially contains *descriptive information* about a system. Compared to systems obser-

vation, systems description describes a systems and its sub-systems themselves (i.e., the inner constitution) instead of describing their behaviors in terms of their input and output. Systems description is often produced by people such as domain experts, designers and engineers of the systems; though it does not have to be. Examples are documents, floor-plans of factories, descriptions of manufacturing processes, maps and satellite images of terrains. Systems description can contain structure and relation information about a system. The analysis and interpretation of systems description often require professional training, domain expertise and knowledge of special notations[5]. Modelers often can not understand or interpret systems description without expert explanations. This type of data has become more and more available, particularly with the increased use of computer-aided and model-based design and engineering.

As stated in §1.2, the objective of this thesis is to provide a method that automatically generates simulation models with flexible structures using existing data. A key construct of the study is to investigate the organization of systems knowledge from the above mentioned sources, and to investigate the embodiment of the knowledge into a method in a formal and flexible way that allows for a class of simulation models to be generated according to different contents of the existing data.

This leads us to follow a systems theoretical approach to simulation modeling. The theoretical framework used as a foundation by this thesis is the systems framework proposed by KLIR and ELIAS (2003) in general systems theory as well as the theory from ZEIGLER et al. (2000) which extends the theory of KLIR and ELIAS (2003) to the context of simulation modeling.

2.1.3 Levels of Systems Knowledge

Epistemology is the theory of knowledge. KLIR and ELIAS (ibid.) presents a hierarchy of epistemological levels of systems. This systems framework organizes distinct levels of systems knowledge in a hierarchy, which is shown in Figure 2.2. The levels also imply the profoundness of our systems knowledge. We give an overview of the levels as following based on KLIR and ELIAS (ibid.).

Level 0 – Source System At the bottom level of the hierarchy is the "primitive understanding" of a system. A source system is defined (according to the modeling interest) by a set of *variables*, a set of potential *states* (values) recognized for the variables, and some operational way of describing the *meaning* of these states in terms of the manifestations of the associated attributes of a real system[6]. A source system is (potentially) a source of empirical data.

Systems at higher levels are distinguished from each other by the level of knowledge regarding the variables of the associated source system. A higher level entails that the

[5] Special notations refer to notations, terminology, rules, *etc.*, used specifically in some organizations, that are unknown to the others.

[6] A variable is an abstract image of an attribute; this means, attributes exist in a real system and variables exist in the model system (KLIR and ELIAS 2003).

knowledge attained at the lower levels is known and it contains additional knowledge that is not available at the lower levels. Hence, the source system is included in all of the higher level systems. Interaction with the world is mediated through the source system to give a data system that is modeled through the levels above.

Level 1 – Data System When the source system is supplemented with data, we view the new system as a data system. The data on the variables are obtained by *measurement* or are defined as *desirable states*. The data obtained by measurement is the systems observation previously discussed. It can describe systems behavior and states. Higher levels involve knowledge of some relational characteristics of the variables defined through which the data can be generated for initial or boundary conditions.

Level 2 – Generative System With knowledge at this level, we are able to define one overall characterization of the constraint among the variables. This characterization describes a process by which the states of the variables (i.e., data) can be generated for each initial or boundary condition. Each variable is defined in terms of a specific translation rule. The rule can be applied either to a variable in the given source system or to a hypothetical variable, introduced for various methodological reasons and usually referred to as an internal variable.

Level 3 – Structure System A structure system is defined as a set of generative systems (or sometimes lower level systems), referred to as sub-systems, that interact with each other in some way.

The status of a system as either a sub-system or super-system is, of course, not absolute. A generative system may be viewed in one context as a sub-system of a structure system, while in another context it may be viewed as a super-system whose sub-systems form a structure system. This *duality* makes it possible to represent each (overall) system by a hierarchy of structure systems[7]. The data type of systems description previ-

Level 4	Meta-Systems	*Relations between relations at Level 3*
Level 3	Structure System	*Relations between models at Level 2*
Level 2	Generative System	*Models that generate data at Level 1*
Level 1	Data System	*Observations and desirable states of Level 0*
Level 0	Source System	*A source of empirical data*

Figure 2.2: Hierarchy of epistemological levels of systems (ibid.)

[7] There are numerous reasons why structure systems are desirable in design and engineering; readers of interest may refer to KLIR and ELIAS (ibid.) or other relevant literature.

ously discussed contains information related to at least generative level.

Level 4 and above – Meta-Systems At level 4 and higher levels, a meta-system or a meta-system of higher orders consists of a set of systems defined at some lower levels and some meta-characterization by which changes in the lower level systems are described. Meta-systems are introduced basically for the purpose of describing changes of systems traits that are defined as *invariant* at lower levels.

Although the hierarchy defined by KLIR and ELIAS (2003) is in the context of general systems theory, it is as well useful for simulation modeling, whose basic assignment is the understanding and representation of systems.

Climbing the epistemological hierarchy often aims at specifying a generative or structure system that can generate correct data. Problems of this nature fall into the category of *systems modeling*; this category can further be divided into *systems inference* and *systems design* depending on whether the system is in existence (KLIR 1988, ZEIGLER et al. 2000). On the other hand, *systems analysis* is the process of using a generative or structure system to generate data that is at a lower hierarchy level; computer simulation is an example of this type (ZEIGLER et al. 2000).

Systems specification in the context of M&S is basically the formal description of systems knowledge in a specification of a (simulation) model. If we wish to define a simulation model that resembles a system in terms of parts, relations and structures, we should draw a parallel between the levels of systems knowledge and the levels of systems specification. The work of ZEIGLER et al. (ibid.) presents such a parallelism.

2.1.4 Levels of Systems Specification

The levels of systems specification (ibid.) has a similar hierarchy placing emphasis on the M&S context (KLIR and ELIAS 2003). They identify useful ways of describing the systems knowledge at the corresponding levels as given in Table 2.1. The following introduction to the levels is based on ZEIGLER et al. (2000).

Observation Frame The observation frame corresponds to the source system. It specifies which variables to measure and how to observe them over a time base. The time-indexed inputs to systems are called input trajectories; likewise, the time-indexed outputs are called output trajectories. Each pair of input and associated output trajectories is called an input/output (or I/O) pair. The choice of input and output variables depends on our modeling objectives.

I/O Behavior and I/O Function They correspond to the data system. The collection of all I/O pairs gathered by observation is called the I/O behavior of a system. The term I/O function indicates the functional relationships of I/O pairs associated with the *initial states*. For example, given an initial state $q_i \in Q$ or $q_i' \in Q$ of the total state set Q, the

Level	Systems Knowledge	Systems Specification	Validity
4	Meta-System		
3	Structure System	Coupled Component	Structural validity
2	Generative System	State Transition	Structural validity
1	Data System	I/O Function	Predictive validity
		I/O Behavior	Replicative validity
0	Source System	Observation Frame	

Table 2.1: Levels of systems knowledge and systems specification (ibid.)

response of a system to input $\vec{x} \in X$ is output $\vec{y} \in Y$ or $\vec{y}' \in Y$. An initial state and an input trajectory determine a unique output trajectory.

State Transition The state transition of a system corresponds to the systems knowledge at generative level. At this level, we specify how a system changes its states given a current state and an input trajectory so that the correct output trajectory can be generated, i.e., $(q_m \in Q, \vec{x} \in X) \mapsto (q_n \in Q, \vec{y} \in Y)$.

Coupled Component The coupled component corresponds to the structure system in Klir's hierarchy. A component in a model is simply a counterpart of a part (or subsystem) in the original system. A coupled component is a component that specifies the interrelations (hence coupling) of its constituent components. Given the duality of a system as previously discussed, a component can be a coupled component or a component that specifies its state transition.

ZEIGLER et al. (ibid.) does not define a level of systems specification that corresponds to Klir's meta-system. Models at meta-levels are discussed in § 2.2. Before that, we give ZEIGLER et al. (ibid.)'s view on morphism and validity in relation with the levels of systems specification.

Levels of Morphism and Validity In mathematics, a morphism is a map between two mathematical structures in a category. In general, the concept of morphism tries to capture similarity between pairs of systems. The pairs of systems knowledge and specification can be related by morphism at each level of the hierarchy. A morphism at an upper level must imply the existence of morphism at a lower level.

The **validity** of a model is the degree to which the model represents its system counterpart (i.e., morphism) to the extent demanded by the intended use of the model. It refers to the relation between a model, a system and an experimental frame. An **experimental frame** is a specification of the conditions under which the system is observed or experimented with.

The **replicative validity** is affirmed if the behavior of the model, i.e., I/O behavior, and the system agree within acceptable tolerance for all the experiments possible within

the experimental frame. Stronger forms of validity are **predictive validity** and **structural validity**. The predictive validity requires replicative validity and the ability to predict as yet unseen system behavior, i.e., agreement at the I/O function. The structural validity requires agreement at state transition or higher (coupled component). This means that the model not only is capable of replicating the data observed from the system but also mimics it step-by-step and component-by-component the way how the system does its state transitions.

2.2 Meta-Models

There is a necessity for using meta-models when one needs to define a class of models in an abstract way. What are meta-models? Are they meta-systems in the epistemological hierarchy of KLIR and ELIAS (2003)? We try to answer the question in this section.

The etymology of *meta-* is $\mu\varepsilon\tau\alpha$-.

> In ancient Greek and Hellenistic Greek $\mu\varepsilon\tau\alpha$- is combined chiefly with verbs and verbal derivatives principally to express notions of sharing, action in common, pursuit, quest, and, above all, change (of place, order, condition, or nature), in the last sense frequently corresponding to classical Latin words in *trans-* ... English formations with *meta-* meaning "beyond" ... "above, at a higher level" ... These formations became common for scientific terms ... predominantly with the sense "dealing with second-order questions". [8]

This dictionary entry renders important properties of "meta-ness" of models. The term *meta-model* has been in wide use for quite a while. Nevertheless, relevant literature shows alternative interpretations of the term. In the following, we present two distinct roles of models, some different ways of interpretation of meta-models, and then discuss the concept that is used in this thesis.

2.2.1 Roles of Models

Meta-models are models too (VANGHELUWE and DE LARA 2002). The distinction of model roles and corresponding relations between the modeled and the model (i.e., the source and target systems) is essential in understanding the concept of meta-models.

KÜHNE (2006) proposes two fundamentally different model roles by the distinction of tokens and types. A *type* is a general sort of thing and a *token* is a particular concrete instance of a type; types are generally said to be abstract and unique, and tokens are concrete particulars (WETZEL 2011). Applying this distinction to models, **token model**s model particular instances (and their relations), and **type model**s model classes of instances (and their relations).

Example 2.1 *Figure 2.3 illustrates examples of the model roles. The original system S is the highway connections of "Randstad" in the Netherlands.*

[8] Oxford English Dictionary.

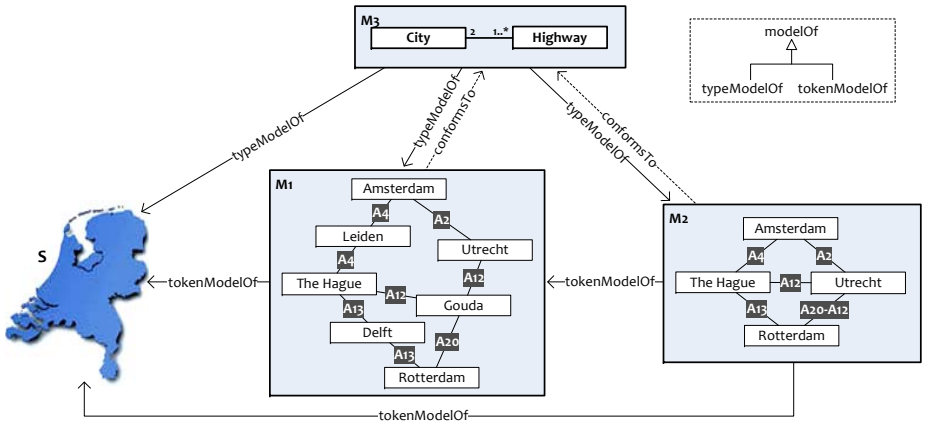


Figure 2.3: Token and type model examples (based on KÜHNE 2006, SPRINKLE et al. 2011)

- *M1 is a token model of S that consists of 7 cities, 8 highway connections and their relations.*
- *M2 is a (simpler) token model of M1 and (thus also) of M2 that consists of 4 cities, 5 highway connections and their relations.*
- *M3 is a type model of S, M1 and M2. M3 consists of 2 types, viz., "City" and "Highway", and their relation. M3 is not a token model of S. Both M1 and M2 conform to M3.*

To summarize KÜHNE (2006)'s propositions of toke and type models:

- A token model captures a certain aspect of the parts and relations in a source system in an one-to-one representation. The abstraction process for creating token models involves no further abstraction beyond projection and translation. A token models are sometimes referred to as snapshot models. They can be used to capture a single configuration of a dynamic system.
- A type model condenses the above one-to-one representations to *concise descriptions* by capturing the types of parts and the relations among the types only. The complete abstraction process for creating type models involves classification in addition to projection and translation. Token models are sometimes referred to as *schema models*, *classification models*, etc. Most models used in model driven engineering are type models.
- The token and type roles are not absolute properties of a model but depend on its relation to its source system.

2.2.2 Types of Meta-Models

A meta-model is a model of models (OMG 2003, SPRINKLE et al. 2011). Although the sentence is weak as a definition, it points out that the "modeling" operation or the `modelOf` relation shall be applied twice (KÜHNE 2006, HESSE 2006). As the `modelOf` relation can be `tokenModelOf` or `typeModelOf`, meta-models can have token or type roles as well. In this section, we show some different types of meta-models that appear in literature. The distinction of the types is based on the model roles from a M&S standpoint.

Meta-Model Type-I The meta-model of this type is a token model of a token model of an original system; that is

$$S_0 \xleftarrow{\texttt{tokenModelOf}} S_1 \xleftarrow{\texttt{tokenModelOf}} S_2$$

where S_0 is the source system of S_1, and S_1 is the source system of S_2. The `tokenModelOf` relation is transitive, *i.e.*, the last token model can reach down to the original system to be its token model in a chain of token models (KÜHNE 2006). The meta-model is therefore also a token model of the original system.

$$S_0 \xleftarrow{\texttt{tokenModelOf}} S_1 \xleftarrow{\texttt{tokenModelOf}} S_2 \Longrightarrow S_0 \xleftarrow{\texttt{tokenModelOf}} S_2$$

In the Example 2.1 previously given, M2 is a Type-I meta-model of S. The works by, *e.g.*, KLEIJNEN and SARGENT (2000) and KLEIJNEN (2009) concern meta-models of this type. By their definition, a meta-model is an approximation of the I/O function that is defined by the underlying simulation model; the meta-model is fitted to the I/O data produced by the experiment with the simulation model. These meta-models are also called response surfaces, surrogates, emulators, auxiliary models, *etc.* (KLEIJNEN 2009)

Meta-Model Type-II The meta-model of Type-II is a type model of a token model of an original system. In this case, the meta-model holds a `typeModelOf` but not `tokenModelOf` relation with the original system.

$$S_0 \xleftarrow{\texttt{tokenModelOf}} S_1 \xleftarrow{\texttt{typeModelOf}} S_2 \Longrightarrow S_0 \xleftarrow{\texttt{typeModelOf}} S_2$$

This means that the meta-model does not model concrete particulars of "a portion of the real world" but a class of those concrete particulars.

In the Example 2.1 previously given, M3 is a Type-II meta-model of S; as such, M3 can model not only S (as M1 and M2 do) but as well any other highway connections. In model driven engineering and development (OMG 2003), the meta-models are of this type. Examples are ATKINSON and KÜHNE (2003) and CETINKAYA and VERBRAECK (2011).

Meta-Model Type-III Recall that the meta-system level of KLIR and ELIAS (2003) describes changes of systems properties that are defined as *invariant* at lower levels. This type of meta-models is very different to the previous two types.

The meta-model represents a larger system (which is a set of lower level systems) and describes a replacement procedure (i.e., replacements of one low level system by another). The meta-system is neither a token model nor a type model of the original system. It is a token model of a larger system that has the original system as a member.

$$S_0 \xleftarrow{\texttt{tokenModelOf}} S_1 \in S_2$$

The works of, *e.g.*, KICKERT and VAN GIGCH (1979) and VAN GIGCH (1991, 1993, 2005) concern meta-models of this type. They discuss meta-models for systems design and organizational decision-making where the meta-model is essentially a model of the modeling or design process.

Discussion The objective of this research is to provide a method that automatically generates simulation models with flexible structures according to the existing data describing different referents within a certain domain, assuming that these simulation models are intended for similar purposes. A simulation model is a token model. It has a concrete referent. Certainly, we do not intend to pre-specify simulation models instance by instance for every referent in AMG. This would be essentially the same as manual processes. In order to specify an abstraction of a class of simulation models within a certain domain, a type model of that domain (with respect to some purposes) is desirable. This means that we need a **Type-II meta-model** for generating simulation models through instantiation by altering model structures and constituent model parts according to existing data.

As stated earlier, a type model models classes of instances and their relations. It captures the types of system parts and the relations among the types only. This requires an appropriate model (and meta-model) representation for the parts and the relations. In this context, we propose component-based models and the use of formal model specifications.

2.3 Component-Based Models

Components are also known as *building block*s. According to VERBRAECK et al. (2002), a **component** is a self-contained, interoperable, reusable and replaceable unit that encapsulates its internal structure and provides useful services to its environment through precisely defined interfaces. Component-based modeling is founded on a paradigm that is common to all engineering disciplines: complex systems can be obtained by assembling components (GÖSSLER and SIFAKIS 2005). It is a promising modeling concept from both systems theory and M&S perspectives.

From A Systems Theoretical Perspective The word *system* in Greek $\sigma\acute{\upsilon}\sigma\tau\eta\mu$-$\alpha$ means *composition* in its literary sense. In § 2.1.1.2, we see that systems can be viewed as parts and relations. A complex system can be recursively decomposed into sub-systems until

an elementary level is reached where we choose to no longer further the decomposition. The criterion for reduction is often functionality (SIMON 1996). SIMON (ibid.) gives a parable of two watchmakers to explain the advantage of using components over using only basic parts. Such a recursive decomposition (or composition when we observe it bottom-up rather than top-down) forms a system into a hierarchical structure.

As stated in § 2.1.3 and § 2.1.4, systems knowledge can be organized into different levels, and we may define the levels of systems specification to correspond with the knowledge levels, which lead to morphisms and validity at different levels (KLIR and ELIAS 2003, ZEIGLER et al. 2000). Together they suggest the use of components and composition to hierarchically represent systems parts and relations.

From An M&S Perspective Modelers use model components to manage complexity in modeling (*i.e.*, for complexity reduction) and to enhance reuse (HOFMANN 2004, MOSER 2006, PAGE 2007, VALENTIN 2011). Thereby, simulation models do not have to be developed from scratch. Components are for composition (SZYPERSKI 2011). The aggregation of simpler components can form more complex components (GÖSSLER and SIFAKIS 2005). Model components can be used as basic building blocks or aggregated building blocks in constructing simulation models (VERBRAECK et al. 2002). Various combinations of model component aggregation bring out different simulation models. ZEIGLER et al. (2000) makes the following comment on components. A component can

(a) be developed and tested as a stand-alone unit,

(b) be placed in a model repository and reactivated at will, and

(c) be reused in any application context in which its behavior is appropriate and its relation to other components makes sense.

An essence of component-based modeling is composability. **Composability** is the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements (PETTY and WEISEL 2003a). It contents with the alignment of concepts by the consistent representation of interpretations of truth in all participating systems (PAGE et al. 2004, TOLK et al. 2013a). An important property that makes models composable is **modularity** (ZEIGLER et al. 2000), i.e., the self-containedness of a model. To be modular, the model components need to fulfill at least two requirements:

(1) they are the outcomes of a meaningful systems decomposition (HOFMANN 2004);

(2) they are expressed by an appropriate modeling formalism (SARJOUGHIAN 2006).

The first requirement is related to *semantics*. Whether the designed model components can be meaningfully composed (PETTY and WEISEL 2003b) is largely determined by whether the original system is meaningfully decomposed. The issue of decomposition is again related to systems knowledge as discussed in § 2.1. Although component-based modeling is widely known and is encouraged for its considerable benefits, it is shown difficult to apply (YILMAZ 2004, SZABO and TEO 2007, TOLK et al. 2010). There is limited literature on, e.g., good ways for systems decomposition (HOFMANN 2004). SI-

MON (1996) introduces the term *near-decomposability* to refer to a property of complex systems which may give us some general idea of decomposition:

(a) the interactions between sub-systems in a complex system are weaker than the interactions within them, and

(b) each sub-system in a decomposed system is almost autonomous, i.e., each has independent functionality but still provides value to the overall system by maintaining a weak connection with it.

The second requirement of applying an appropriate modeling formalism is related to *syntactics*, i.e., whether the components can be correctly connected, which requires that the components are constructed such that their parameter passing mechanisms, external data accesses, etc., are compatible for all of the different configurations that might be composed (PETTY and WEISEL 2003b).

Additionally, an appropriate modeling formalism can support the component-based modeling concept. This means that it can support a clear definition of components, aggregated components and their composite relations, and it can support the construction of models through compositional variations of components (NIERSTRASZ and MEIJLER 1995, NIERSTRASZ and TSICHRITZIS 1995, ACHERMANN et al. 2001). One can achieve this by a careful choice of modeling formalisms which are discussed next.

2.4 Modeling Formalisms and Model Specifications

A *simulation model* is a special piece of software. It is a set of instructions, rules, equations, or constrains for generating I/O behavior; see § 2.1.4. There are many constrains on how it should be designed and developed. These constrains partly concern the morphisms between the systems knowledge and the systems specification as discussed in § 2.1.4. The constrains also concern the non-functional requirements for the model and the model components, e.g., composability, modularity, reusability and extensibility.

Systems specifications need to be expressed in a certain modeling formalism. A formalism has the advantages that it has a sound mathematical foundation and it has rigorously defined semantics (ZEIGLER et al. 2000). A formalism can largely influence model design. We next present basic classes of modeling formalisms and give the rationale for our formalism choice.

2.4.1 Formalisms and Formalism Classes

Modeling formalisms are also known as systems specification formalisms. They are shorthand means of delineating a particular system within a subclass of all systems (ibid.). In this context, a formalism is mathematical[9]. More specifically, a **modeling**

[9] A *mathematical formalism* refers to a — mathematically formulated and physically interpreted — notational system of locally applicable rules that derive from (but need not be reducible to) fundamental theory (GELFERT 2011).

formalism is a mathematical structure[10] composed of sets, relations on sets, and axioms on relations, for expressing simulation models.

2.4.1.1 Three Classes of Formalisms

There is a variety of modeling formalisms in M&S literature, e.g., differential algebraic equations, bond graphs, cellular automata, petri nets, just to name a few. Figure 2.4 shows a subset of modeling formalisms where many commonly used formalisms are included. The arrows denote transformation relations that are discussed in § 2.4.2. To give an overview of the formalisms, we do not discuss each in detail but in terms of classes following ZEIGLER et al. (2000). There is a vertical dashed line in the middle of Figure 2.4 that separates the formalisms into two classes: formalisms for discrete models on the right side, and formalisms for continuous models on the other side.

In traditional mainstream M&S paradigms, (dynamic) systems are delineated with *discrete or continuous time*. This means that the state changes of the systems are specified on a discrete or continuous time base[11]. The resulting models are *discrete time*

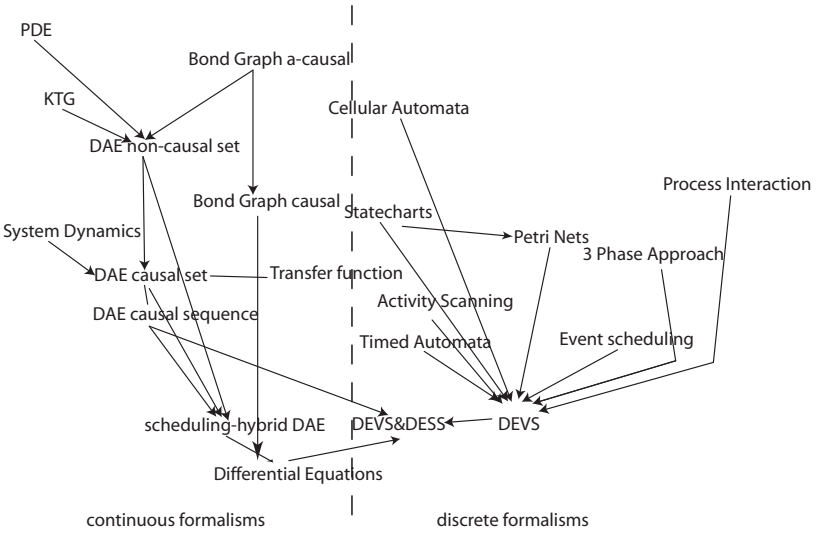


Figure 2.4: Formalism Transformation Graph (VANGHELUWE 2000, modified by JACOBS 2005)

[10] A *mathematical structure* is a generic name for unifying concepts whose general characteristic is that they can be applied to sets whose elements are of an indefinite nature; in order to define a structure, relations are given in which the elements of the set appear (the type characteristic of a structure), and it is then postulated that these relations satisfy certain conditions — axioms of the structure (EOM 2011).

[11] A time base is defined as a structure $time = (T, <)$ where T is a set and $<$ is a transitive, irreflexive, and antisymmetric ordering relation on elements of T, i.e., $\forall t \in T$ (ZEIGLER et al. 2000). In M&S, time is conceived

models or *continuous time models*. As the continuous time models also have continuous states, we can simply call them *continuous models* without any ambiguity in meaning. The formalisms used to specify these models are **discrete time formalisms** or **continuous formalisms** respectively. These are the two traditional classes of modeling formalisms. Examples are formalisms for difference equations and differential equations.

There is a third class of modeling formalisms, the **discrete event formalisms**, that is "the new guy on the block" using ZEIGLER et al. (ibid.)'s expression. In fact, the discrete event world view is rather natural and intuitive in many situations. Consider the following simple case.

Example 2.2 *My phone was ringing when I was home one evening and I picked up the call. It was a friend asking if we could meet in an hour in the city. I was free that evening so I agreed to meet. As I would need 20 minutes to get to the city, I left home in 40 minutes.*

> **Remarks** The actions in the example may be roughly segmented into two: (i) picked up the call and agreed to meet a friend, and (ii) left home. The first action was activated by an external event of "phone was ringing" by which the input value of the event was "a friend asked if we could meet". The response was determined by the input value and my (internal) state of being "free that evening". As a consequence, the second action was scheduled in 40 minutes (i.e., an internal event) which was deduced by the meeting time and the travel time.

The discrete event world view brought up a different modeling paradigm in which the states (and actions, i.e., state-to-output mapping) of a system are determined by external and internal events. Despite the increased use of this world view in practice, the modeling approaches based on this world view were not formalized until a few decades ago (HO 1994) — much later than the former two classes of formalisms. An example of a discrete event formalism is *petri nets*.

2.4.1.2 Three Basic Formalisms

The three classes of **discrete time**, **continuous**, and **discrete event** models and formalisms are not unrelated, neither are they mutually exclusive. Their relations are particularly shown by the work of ZEIGLER (1976) and ZEIGLER et al. (2000) where three basic modeling formalisms (and some variations) are developed, each of which corresponds to one of the three classes of formalisms just discussed. The three formalisms are:

- *Discrete Time System Specification* (DTSS),
- *Differential Equation System Specification* (DESS), and
- *Discrete Event System Specification* (DEVS).

They together form a unified systems theory based M&S framework. We next shortly present the three basic modeling formalisms and then discuss their relations.

---

as flowing along independently, and all dynamic changes are ordered by this flow (ibid.).

Discrete Time System Specification (DTSS) for Discrete Time Models This formalism represents systems over a discrete time base. It assumes a stepwise execution. At a particular instant the model is in a particular state and it defines how this state changes, i.e., what the state at the next instant will be. If the state at time t is $q(t)$ and the input at time t is $x(t)$, then the state at time $t+1$ is $q(t+1) = \delta(q(t), x(t))$ where δ is the state transition function.

Differential Equation System Specification (DESS) for Continuous Models This formalism represents systems with continuous state over a continuous time base. It does not specify a next state directly through a state transition function but specifies the rate of change of the state variables $q_{i\in[1,n]}$ through a derivative function f. This means that at any particular instant, given a state and an input value, we know the rate of change of the state, *i.e.*, $dq_{i\in[1,n]}(t)/dt = f(q_1(t), q_2(t), \cdots, q_n(t), x(t))$, and can thus compute the state at any instant in the future using integration methods.

Discrete Event System Specification (DEVS) for Discrete Event Models This formalism represents systems as piecewise constant state trajectories over a continuous time base. The state trajectories are produced by state transition functions δ_{ext} and δ_{int} that are activated by external or internal events. More details of the formalism is presented in § 2.4.3.

The other formalisms presented by ZEIGLER et al. (2000) are in principle variations and combinations of the three basic formalisms. The choice of the underlying formalism for systems specifications is often based on, e.g., the nature of the M&S problem and the modeler's experience and background.

2.4.1.3 Our Choice of Modeling Formalisms

In this research, we choose DEVS as the underlying modeling formalism. Our choice is mainly based on two criteria.

- DEVS provides strong support for hierarchical component-based modeling, and
- DEVS can embed and represent many other formalisms.

The strong support of the DEVS formalism for hierarchical component-based modeling can be shown by the formalism itself (which is soon presented in § 2.4.3) and by the applications of the formalism (WAINER 2009). A basic motivation of discrete event modeling in general is that it is intrinsically tuned to the capabilities and limitations of digital computers (ZEIGLER et al. 2000). DEVS supports port-based modeling which is proposed by a number of authors for its modular principle (PAREDIS et al. 2001, LIANG and PAREDIS 2004, BREEDVELD 2009). In general, the interface of port-based models only consists of input and output ports, through which all interactions with the environment occur. Thereby, direct dependencies between components are largely eliminated (RÖHL and MORGENSTERN 2007).

That DEVS can embed and represent other formalisms is argued by authors such as DE LARA et al. (2004), VANGHELUWE (2000, 2008) and WAINER (2009). They show that DEVS is a fairly general formalism and can play the role as a root formalism within the class of discrete event formalisms. Additionally DEVS can be used to represent or approximate models using discrete time and continuous formalisms. These issues are explained in the following based on ZEIGLER et al. (2000) and VANGHELUWE (2000, 2008).

2.4.2 Formalism Transformations

DEVS Embedding The DEVS formalism has a fairly general form that underlies systems with discrete event behavior. As such, many other discrete event formalisms can be embedded as sub-formalisms of DEVS. This is called *DEVS embedding*. According to ZEIGLER et al. (2000), one formalism can be embedded in another if any system in the first can be simulated by some system in the second. The formalism embedding relations (i.e., transformation relations from other discrete event formalisms to DEVS) are shown by the arrows on the right half side of Figure 2.4.

DEVS Representation The DEVS formalism supports the specification of all three basic classes of models discussed in § 2.4.1. In other words, DEVS can also be used to represent models specified by discrete time formalisms and continuous formalisms. This is called *DEVS representation*. The formalism representation relations (i.e., transformation relations from discrete time and continuous formalisms to DEVS) are shown in Figure 2.4 by the arrows pointing from the left hand side to DEVS.

As depicted in Figure 2.4, the formalism transformations can eventually converge to the DEVS formalism. This confirms that DEVS is a general modeling formalism. VANGHELUWE (2000) argues that DEVS is a common denominator of modeling formalisms as it unifies continuous and discrete representations.

Demonstrated by ZEIGLER et al. (2000), DEVS can strongly represent the DTSS formalism by constraining the time advance to be constant. This strong representation embeds DTSS in DEVS. Since continuous models are traditionally simulated in discrete time, the embedding of DTSS in DEVS offers an immediate path to the DEVS representation of DESS (approximations) through *discretization* (of time). Approximation can, however, also be achieved through *quantization* (of state variables).

Discretization and Quantization Discretization refers to the more traditional way of discretizing the time base in order to represent continuous systems with approximation. ZEIGLER (1998) proposes the theoretical foundation of an alternative approximation approach in M&S, namely quantization, by which the state variables of continuous systems are discretized (or partitioned) rather than the time base. Discretization leads to discrete time models while quantization leads to discrete event models. The work of ZEIGLER et al. (2000) also shows that quantized DEVS representation can provide more efficient simulation at lower error cost than the traditional integration methods of differential equation systems.

2.4.3 DEVS Formalism

The DEVS formalism specified by ZEIGLER et al. (2000) exhibits the concepts of systems knowledge and specification as discussed in § 2.1.3 and § 2.1.4, and supports hierarchical component-based modeling as discussed in § 2.3.

2.4.3.1 General Description

It is mentioned earlier that DEVS models are port-based. The interfaces of the models only consist of input and output **ports** , through which all interactions take place. The interaction relations of the models are defined by **couplings** where the model ports are connected from one to another. A DEVS model can place a value on one of its ports but the actual destination of the output is not determined until the model becomes a component in a larger model and the coupling relation is specified (ibid.). Composition in DEVS is achieved through couplings. DEVS models can be coupled together (when appropriate) to form a larger model component which is called a **coupled model**. The ability to incrementally compose larger coupled models from previously constructed components leads to hierarchical model construction. When we view the hierarchical composition as a tree structure, the nodes in the tree are the coupled models whereas each leaf in the tree is an **atomic model**. An atomic model is not a resultant of composition. It generates the behavior of a system at an elementary level. More specifically, an atomic model possesses states and state transition mechanisms (dictating how inputs transform current states into subsequent states) as well as the state-to-output mapping (ibid.). Hierarchical DEVS models are coupled models with components which may be atomic or coupled models. Atomic models are expressed in the basic DEVS formalism; coupled models are expressed in the DEVS coupled model formalism providing component and coupling information. The following description of the formalism is based on ZEIGLER et al. (ibid.).

2.4.3.2 Basic DEVS Formalism for Atomic Models

A basic *discrete event system specification* is a structure

$$M = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta) \tag{2.3}$$

where

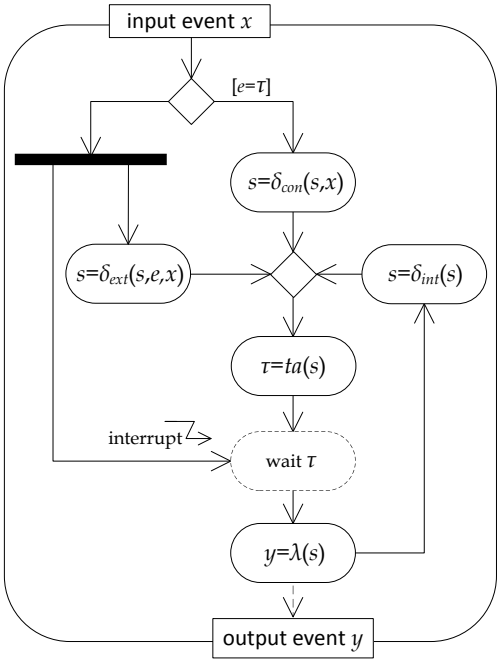
$X = \{(p, v) \mid p \in IPorts, v \in X_p\}$	is the set of *input ports* and *values*
$Y = \{(p, v) \mid p \in OPorts, v \in Y_p\}$	is the set of *output ports* and *values*
S	is the set of *sequential states*
$\delta_{ext} : Q \times X^b \to S$	is the external state transition function, where
	$Q = \{(s, e) \mid s \in S, e \in [0, ta(s)]\}$ is the set of *total states*
	e is the *elapsed time* since last transition
$\delta_{int} : S \to S$	is the internal state transition function
$\delta_{con} : Q \times X^b \to S$	is the confluent state transition function

$\lambda : S \rightarrow Y^b$ is the output function
$ta : S \rightarrow \mathbb{R}_0^+ \cup \infty$ is the time advance function

The superscript b in the specification indicates a bag of input or output values in stead of a single value. A **bag** is a set with possible multiple occurrences of its elements. The interpretation of the formalism is illustrated in Figure 2.5. For simplicity, we first discuss the state transition mechanism without the confluent state transition function $\delta_{con}(s,x)$.

In principle, an atomic model can change its state $s \in S$ by the **external state transition function** $\delta_{ext}(s,e,x)$ and the **internal state transition function** $\delta_{int}(s)$. The external state transitions are determined by s, $e \in [0,\tau]$ and $x \in X^b$ where e is the **elapsed time** since the last external or internal state transition[12], and x is the input which can have a bag of values[13]. The **time advance function** $ta(s)$ determines the designated **life time** $\tau \in \mathbb{R}_0^+ \cup \infty$ of a state s. It is the time a certain state is supposed to last. A time advance is determined right after an external or internal state transition.

An external state transition is activated by an *external event* (or input event), i.e., each time when there is an input placed on one of the input ports. The internal state



►*internal transition first (default):*
$\delta_{con}(s,x) = \delta_{ext}(\delta_{int}(s), 0, x)$

►*external transition first:*
$\delta_{con}(s,x) = \delta_{int}(\delta_{ext}(s,\tau,x))$

Figure 2.5: State transition mechanism of a DEVS atomic model

[12] This means that the elapsed time e starts to count from zero right after each transition.
[13] The values are often associated with ports which are not shown in Figure 2.5.

transitions are self-triggered (or time-triggered). This means that when the elapsed time reaches the life time of a state, referred to as an *internal event*, an internal state transition is activated. Right before the internal state transition, based on the present yet right-to-be-changed state, the **output function** $\lambda(s)$ places output values $y \in Y^b$ on one or more output ports.

As just mentioned, an external state transition is set off when an external event arrives. This entails that an external event causes an external transition which further causes a reset of elapsed time, hence also interrupts a "planned" internal transition. What happens when an external event arrives right at the time of an internal event? This is where the **confluent state transition function** $\delta_{con}(s,x)$ comes into play. When there is a collision between external and internal events, the confluent state transition decides the next state. In general, the modeler is unrestricted in defining the confluent state transition function. There are, however, two commonly used definitions. The default definition is to have an internal transition followed by an external transition, i.e., $\delta_{con}(s,x) = \delta_{ext}(\delta_{int}(s),0,x)$; another possibility is to have an external transition followed by an internal transition, i.e., $\delta_{con}(s,x) = \delta_{int}(\delta_{ext}(s,\tau,x))$.

The basic DEVS formalism corresponds to systems knowledge at the generative level where state transitions of systems can be specified (§ 2.1.3 and § 2.1.4). Taking into account the duality of generative and structure systems discussed in § 2.1.3, some may view the basic formalism as being at the structure level of the knowledge hierarchy. The basic DEVS formalism is consistent with the definition of systems by WYMORE (1967), § 2.1 Eq. 2.1, and can be considered as a refinement or extension of the definition (ÖREN and ZEIGLER 2012).

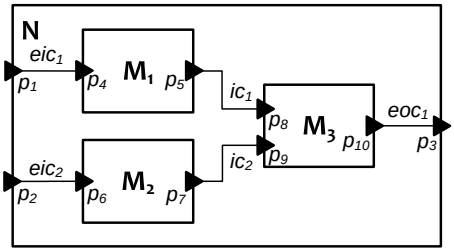
2.4.3.3 DEVS Coupled Systems Specification Formalism for Coupled Models

A DEVS coupled model is also a structure

$$N = (X, Y, D, \{M_{d\in D}\}, EIC, EOC, IC) \tag{2.4}$$

where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$	is the set of (external) *input ports* and *values*
$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$	is the set of (external) *output ports* and *values*
D	is the set of *component names*
$\{M_{d\in D}\}$	is the set of *components*, each of which can be an atomic model or a coupled model
$EIC \subseteq X \times \cup X_{d\in D}$	is a set of *external input couplings* that connects external inputs to component inputs, where $X_{d\in D}$ is the set of inputs of $M_{d\in D}$
$EOC \subseteq \cup Y_{d\in D} \times Y$	is a set of *external output couplings* that connects component outputs with external outputs, where $Y_{d\in D}$ is the set of outputs of $M_{d\in D}$
$IC \subseteq \cup Y_{d\in D} \times \cup X_{d'\in D \wedge d\neq d'}$	is a set of *internal couplings* that connects component outputs to component inputs



$IPorts = \{p_1, p_2\}, OPorts = \{p_3\}$
$\{M_{d \in D}\} = \{M_1, M_2, M_3\}$
$IPorts_1 = \{p_4\}, IPorts_2 = \{p_6\}, IPorts_3 = \{p_8, p_9\}$
$OPorts_1 = \{p_5\}, OPorts_2 = \{p_7\}, OPorts_3 = \{p_{10}\}$
$EIC = \{eic_1 = (p_1, p_4), eic_2 = (p_2, p_6)\}$
$EOC = \{eoc_1 = (p_{10}, p_3)\}$
$IC = \{ic_1 = (p_5, p_8), ic_2 = (p_7, p_9)\}$

Figure 2.6: DEVS coupled model: an example

A coupled model represents a sub-system at a non-elementary level. It is a resultant of composition. This means that a coupled model does not generate behavior per se, as that of an atomic model, but it specifies its constituent **components** and their **coupling relations**. The components are systems specifications *on their own*. A component in this context may be an atomic model specified using the basic DEVS formalism (Eq. 2.3) or a coupled model specified using the DEVS coupled model formalism (Eq. 2.4). The behavior of a coupled model is determined by the collective behavior of its components.

Couplings in DEVS can be of three types. The couplings from (the input ports of) the coupled model to (the input ports of) its components or from (the output ports of) its components to (the output ports of) the coupled model are the **external input couplings** (EICs) and **external output couplings** (EOCs) respectively. The coupled model and its components are also called the **parent model** and the **child models**. The couplings from (the output ports of) a child model to (the input ports of) another child model are the **internal couplings** (ICs).

In DEVS, self-couplings (or direct feedback loops) are impermissible, i.e., a model's output ports can not be directly connected to its own input ports. One-to-many ($1 : n$) and many-to-one ($n : 1$) couplings are possible in DEVS. (The indexes refer to ports.) For example, in specifying ICs, a single output port may be coupled to multiple input ports, and similarly multiple output ports may be coupled to a single input port.

Figure 2.6 gives a simple example of a DEVS coupled model. The coupled model N consists of three components $M_{d \in \{1,2,3\}}$. Each component may be specified by Eq. 2.3 or Eq. 2.4. The coupled model also specifies five coupling relations, two EICs, one EOC, and two ICs. The example only shows one-to-one couplings. However, it is possible to have, e.g, the couplings ic_1 and ic_2 both connect to p_8 or p_9. Couplings such as connecting p_{10} to p_8 or p_9 are not allowed.

The DEVS coupled systems specification formalism corresponds to systems knowledge at the structure level where a set of systems and their interrelations can be specified (§ 2.1.3 and § 2.1.4). It is apparent that varying the coupling relations of the child models can alter the behavior of their parent model. In such cases, the parent model has been changed (to be another model) as its structure has been changed to another.

2.4.3.4 Remarks On DEVS Model Composition

The structure change at the coupled model level leads to different model composition. It is a useful model generation strategy because it links to the essence of design as understood in systems theoretical terms. In ZEIGLER et al. (2000), systems design is understood as the process of going from a lower to a higher level of systems epistemology, namely from the generative (or lower order level) systems to that of a higher order level system (§ 2.1.3 and § 2.1.4).

Modelers use components, with the knowledge of their inner structure and outer behavior as his or her basic vocabulary, and connect the components within new structures, encoding new knowledge that was not available in the previous step. The result of this modeling activity can in turn be analyzed by simulating it, which corresponds to automatically translating the structure system to a data system (§ 2.1.3).

With the *closure under coupling* property[14] of DEVS, the resultant of couplings of DEVS models is also a DEVS model, making it possible to incrementally develop a hierarchy of more and more complex model components, whose patterns of state changes could not have been specified all at once, or the modeler choose not to specify them all at once.

Furthermore, the hierarchical composition of models makes it possible to separate the different levels of organization which are relevant for systems design and systems analysis (§ 2.1.3). This makes model understanding easier as long as the world view guiding the component (library) specifications coincide with the world view of the end users, i.e., when the morphisms between the systems knowledge (of the users) and the systems specifications at different levels are respected (§ 2.1.2 and § 2.1.4).

There is, however, a challenge in hierarchical composition of models: model components must be developed in the first place. This development requires a top-down hierarchical decomposition of the system (or the class of systems) under modeling interest, in order to identify the different parts and relations relevant in the domain of application. As mentioned in § 2.3, there is a limited number of literature on scientific methodologies for systems decomposition for model component development (HOFMANN 2004). Decomposition in practice is often done according to a certain functional perspective, which per definition excludes many aspects of the system. This will result in certain components that can not be used to compose a useful model in some other context. Therefore, the decomposition strategy must be strongly communicated to the end users of the model component library.

[14] A modeling formalism has the property that it is *closed under coupling* if the resultant of couplings of systems specified in this formalism is itself a system specified in the formalism, i.e., the resultant defines a basic system in the same formalism; closure under coupling allows us to use couplings of systems as components in a larger coupled system, leading to hierarchical, modular construction (ZEIGLER et al. 2000).

3

An Outlook on Automated Simulation Model Generation

OUR STANDPOINT OF AMG is within a certain domain of application. As discussed in § 1.1, many organizations are facing the situation that they often need simulation studies where the models take a long time to develop. Since more and more data has become available in organizations, it could provide useful information for modeling. But constrained by time and cost, much of the data is currently unused for modeling. To improve the situation and benefit from the data, there is a need for a method that can use existing data to automatically generate simulation models.

The method proposed by this research is presented throughout § 4, § 5 and § 6. In this chapter, we intend to provide an outlook on AMG as well as the content of the next three chapters. In § 3.1, we discuss the research questions in § 1.2 and identify a set of constructs for the AMG method based on the concepts and theories presented in § 2. The relations between the domain, model, research questions and constructs are illustrated.

To elucidate data quality issues that one may encounter in using existing data for

AMG, a modeling example is given in § 3.2 to show the difference between the information contained in the existing data and that of the a simulation model. In § 3.3, we discuss data quality categories and criteria that are pertinent to the context of this research.

Starting from existing data to simulation models, we propose to divide an AMG process into three steps. These steps and their relations to the proposed constructs are briefly discussed in § 3.4.

3

3.1 Proposed Constructs on Research Questions

In this section, we discuss the research questions (RQs) specified in § 1.2 and propose a set of *constructs*[1] that can be used in an AMG method. The identification of these constructs is a first step towards tackling the research problem. Recall that our research objective is to provide a method that automatically generates simulation models using existing data. Given a class of potential referents within a domain of application and existing data about the referents, we shall find a method that can automatically generate simulation models of the referents according to the data.

On Research Question 1

What is a good way to define flexible structures for simulation models in order to achieve the research objective?

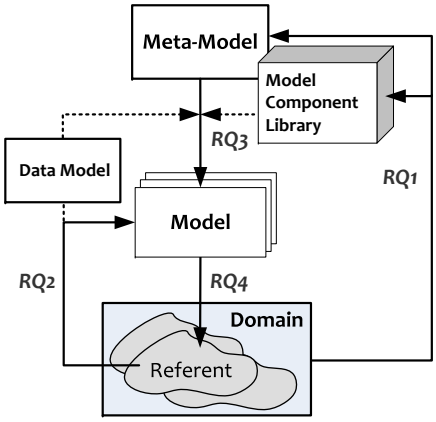
RQ1 is concerned with simulation model structures. As argued in § 1.1, we aim at generating simulation models whose structures are not pre-defined (before model generation). A simulation model is a token model that has a concrete referent in a domain of interest. Because pre-defining structures on concrete simulation model instances is inefficient, inflexible and sometimes impossible, we need to specify an abstraction of the structures of a class of simulation models within a certain domain. Following the discussion in § 2.2, a Type-II meta-model of the domain is desirable as a basis for generating the desired simulation model (structure) through instantiation.

Construct 1 *A Type-II meta-model (i.e., a type model) of the systems of interest.*

As indicated by the RQ1 arrow in Figure 3.1, starting from a class of potentially infinite but similar modeling problems (i.e., source systems or referents) within a domain, we shall define a Type-II meta-model that provides an abstraction of the structures of the potential simulation models.

The discussion in § 2.2.1 points out that a type model captures the types of system parts and the relations among the types only. It is a necessary but yet insufficient modeling basis for AMG. We still need appropriate modeling representations for the parts,

[1] *Constructs* are representations of the entities of interest in a theory; they are the basic building blocks or components of a theory (DUBIN 1978, GREGOR and JONES 2007, KUECHLER and VAISHNAVI 2008). We use the term *construct* instead of *building block* or *component* in order to differentiate it from the term *model component*.



RQ1 What is a good way to define flexible structures for simulation models in order to achieve the research objective?

RQ2 What are the requirements for the data in order to achieve the research objective?

RQ3 What functionalities should a method provide in order to automatically generate simulation models with flexible structures using existing data?

RQ4 What is the quality of the simulation model generated by the method?

Figure 3.1: Research questions in relation with domain, model and meta-models

and a *convenient* way to assemble the model parts and to set up their relations. In this regard, we propose the use of hierarchical component-based modeling and an appropriate underlying modeling formalism, for the reasons presented in § 2.3 and § 2.4. A model component library can be developed to fulfil these conditions.

Construct 2 *A model component library that implements the concept of hierarchical component-based modeling and uses an appropriate underlying modeling formalism.*

Although we choose and recommend the DEVS formalism because of its fair generalness and its other characteristics reasoned in § 2.4, modelers could best decide on the formalism that suits the nature of the M&S problems.

On Research Question 2

What are the requirements for the data in order to achieve the research objective?

RQ2 is concerned with data issues. When we have an appropriate domain meta-model and a suitable model component library as just proposed, how to generate simulation models from them? We need *concrete information* on how to instantiate a simulation model, i.e., the information about the *model structure* and the *model parameterization*. This means that we need to have good data. For the research objective, good data would be those that could provide or potentially provide sufficient information about the model structure and parameterization for the instantiation of simulation models given the meta-model and the model component library.

This suggests that RQ1 and RQ2 are not independent of each other. The existing data is obtained from a referent or it describes a referent. We call them systems observation and systems description in § 2.1.2. If the data is to give certain information about the model of the referent, we logically want to create a parallelism between the referent and its model. This parallelism can be achieved by creating morphisms between

the model component hierarchy and different systems knowledge levels as discussed in § 2.1.3 and § 2.1.4. Accordingly, the meta-model and the model component library should represent corresponding systems knowledge levels. Hence, we revise the first two constructs as follows.

Construct 1 (Revised) *A Type-II meta-model (i.e., a type model) that represents corresponding knowledge levels of the systems of interest.*

Construct 2 (Revised) *A model component library that conforms to the domain meta-model, implements the concept of hierarchical component-based modeling, and uses an appropriate underlying modeling formalism.*

When the domain meta-model and the model component library are designed to support the morphisms at different levels, the existing data can be used to provide the information about the model structure and parameterization. As explained in § 1.1 and § 1.2, data that is (appropriately) specified for AMG would contain sufficient information ready for AMG; while in the existing data, the information of the right content and structure is often not readily available, i.e., the data has quality issues. We need to find out what information the existing data contains or could contain, what information is necessary, and how to cover the information gap. This relation is indicated by the RQ2 arrow in Figure 3.1.

The existing data needs to be transformed (or rewritten) in order to obtain a representation according to which a simulation model can be generated. In this regard, the transformation rules shall be specified. The data is obviously on concrete referents in a domain. Therefore, we need a meta-model of the data, i.e., a data model, on which the transformation rules can be specified.

Construct 3 *A data model of the existing data of interest; or multiple data models if the existing data has multiple data sources.*

According to the discussion in § 2.2, the data model is a type model, a Type-II meta-model of the existing data (as the data can be seen as a model of a referent).

On Research Question 3

What functionalities a method should provide in order to automatically generate simulation models with flexible structures using existing data?

Built on the previous two RQs, RQ3 is concerned with the AMG method itself. Given a data model of the existing data, a domain meta-model and a suitable model component library as just discussed, the goal is to generate a simulation model according to the existing data. The method is bound to model transformation. The transformation definition should be specified on the meta-models (see § 3.4.1), i.e., from the data model to the domain meta-model; and the transformations are performed on concrete (model) instances, i.e., from the existing data with the (instantiations of) model components to the simulation model.

Construct 4 *A model transformation definition specified on the data model(s) defined by Construct 3 and on the Type-II meta-model defined by Construct 1.*

These relations are indicated by the RQ3 arrow and the dotted arrows in Figure 3.1. It should be clear that the more information gap the data model and the Type-II meta-model have, the more complex the model transformations would be.

The AMG method should be able to perform a full instantiation of the simulation model (including model structure and model parametrization). The model transformation rules, if appropriately specified, shall be able to instantiate model structures. Model parameterization is possible when the parameters are the observables. When some parameters are unobservable, they can be configured manually or with default values. We may wish to automatically calibrate these parameters if the empirical data exists. In such cases, we need an automated procedure for model calibration.

Construct 5 *A procedure for model calibration after initial generation of simulation models when empirical data of the systems of interest exists.*

This construct is proposed as a complement to the AMG method. The subject of model calibration, however, is not a major focus of this thesis.

On Research Question 4

What is the quality of the simulation model generated by the method?

RQ4 is concerned with the quality of the simulation model generated. The quality of the simulation model reflects the quality of the AMG method. Hence, RQ4 is also concerned with the evaluation of the AMG method. The most important quality of a simulation model is its validity, a relation indicated by the RQ4 arrow in Figure 3.1. Model validity is not an issue that only comes into sight after the overall simulation model is built. Discussed extensively in M&S literature, it is an outcome of joint efforts of design, test, verification and validation, among others, at each model development stage. We use the arrow between the model and the referent simply to indicate the last stage of validation. In this research, the final model validation is organized with an expert panel.

We summate the five constructs proposed.

1. **Domain Meta-Model** A Type-II meta-model (i.e., a type model) that represents corresponding knowledge levels of the systems of interest.
2. **Domain Model Component Library** A model component library that conforms to the domain meta-model, implements the concept of hierarchical component-based modeling, and uses an appropriate underlying modeling formalism.
3. **Data Model** A data model of the existing data of interest; or multiple data models if the existing data has multiple data sources.
4. **Model Transformation** A model transformation definition specified on the data models defined by Construct 3 and on the Type-II meta-model defined by Construct 1.

5. Model Calibration A procedure for model calibration after initial generation of simulation models when empirical data of the systems of interest exists.

Hereafter, when we use the term *meta-model*, it only refers to *Type-II meta-models* if not otherwise indicated.

3.2 A Modeling Example

To make further discussion more concrete, we give a modeling example.

3.2.1 Vignette: A Tram Crossing

Example 3.1 *Suppose we need to simulate a tram crossing to study trams' driving time and waiting time. The infrastructure at the crossing is given by a CAD drawing (in metric) as shown by Figure 3.2. The lines represent the rail tracks. Each orange labeled circle (source#) indicates a stop. Each red labeled circle (H#) indicates where the trams are to be generated. The "rules" of the model are of common sense:*

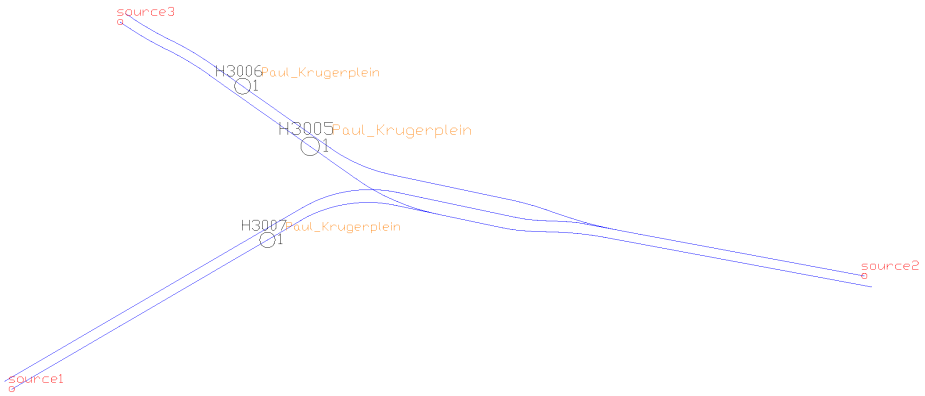
1. *Trams shall be generated with service line numbers (SLNs) at scheduled time according to given timetables.*
2. *A tram shall drive in a realistic manner along a designated SLN route and it shall halt at each stop along the route according to a given halting time distribution.*
3. *A realistic manner means, e.g., trams wait for one another when necessary and they drive with acceleration and deceleration respecting traffic rules.*
4. *When there is a conflict at the crossing, trams shall pass according to standard priority rules.*

The following four types of data are provided: (i) a CAD drawing of the rail infrastructures, (ii) timetables, (iii) service routes, and (iv) halting time distributions.

What is a good method to automatically generate a simulation model from the data taking into account that the data can change its content and scale?

We do not intend to propose a solution in this section but to elucidate some data quality issues that we may encounter in AMG or in modeling in general. For AMG, it is reasonable to first ask "what need to be transformed into what" (MENS and VAN GORP 2006). Not until we make clear the input and output of a transformation and their differences, could we design and specify a transformation that should leverage these differences. The first what asks about the syntactics, semantics and pragmatics of the data, i.e., what information the data directly or indirectly conveys. The second what asks about the simulation model. More specifically, in the case of component-based modeling proposed by Construct 2, we need to know what model components are needed, what composition they have, and how to configure them.

It can be convenient to consider the input and output of a transformation in a reversed order. Therefore, we next first discuss the components and composition in the



The CAD Dataset:

```
Entity 1: line, start point, end point, color, ...;
Entity 2: line, start point, end point, color, ...;
Entity 3: arc, start point, end point, center, radius, ...;
Entity 4: circle, center, radius, color, name, ...;
                        ...
Entity n: line, start point, end point, color, ...;
```

Figure 3.2: A CAD drawing of the tram crossing

simulation model of the example, and then compare them with the data available on hand.

3.2.2 Model Components and Composition

The AMG method is intended for a class of potential referents in a certain domain. As such, we should design a model component library that provides sufficient flexibility and generality to accommodate a predefined scope of possible changes of referents. This can be achieved, among others, through alteration of model composition and parameterization. We leave further discussion about model component library for § 4, and for now assume that the required components for Example 3.1 are readily available. We use the terms *atomic model* and *coupled model* in the DEVS formalism referring to the model components at elementary and non-elementary levels. For Example 3.1, these components are:

1. source (SR) ⟶ coupled: vehicle generator,
2. vehicle generator (VG),
3. sink (SK),
4. vehicle (V),
5. track (T),

6. stop (ST) ⟶ coupled: track, sensor,
7. sensor (S),
8. switch (SW),
9. traffic light (TL),
10. control unit (CU), and
11. crossing (CR) ⟶ coupled: track, switch, traffic light, control unit.

The components 1, 6, and 11 are coupled models whose component members are indicated. The other components listed are atomic. Using these components, we may obtain a model composition whose schema is shown by Figure 3.3. Each circle in the schema corresponds to a model component instance — circles with thin lines for atomic models or thick lines for coupled models. For simplicity of the illustration, the labeled ("tracks") arrows do not represent direct coupling relations but a sequence of "track" component instances coupled one after another in which the first track and the last track are coupled to the component instances indicated. The "tracks" arrows denote the directions of the traffic flow. The schema on the left of Figure 3.3 gives a top level view of the model composition. The two smaller ones on the right show the inner structures of the two coupled models "stop" (ST) and "crossing" (CR) respectively.

The "vehicle" instances (i.e., trams, not shown in the schema) are dynamically generated during a simulation run by the "vehicle generators" in the "sources" (SR) according to the timetables. They then drive on the "tracks" along designated routes. Their driving behaviors, e.g., the last three rules specified in Example 3.1 (p. 40), are encoded in the "vehicle" component (see § 4.3.3). Once they reach the "sink" (SK) component instances, the "vehicles" are removed from the simulation model.

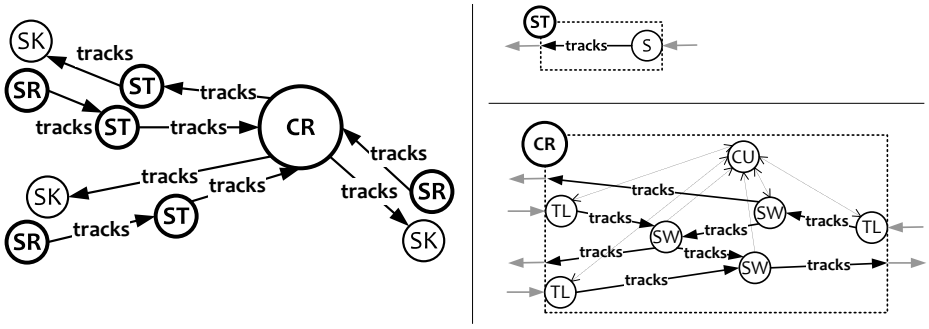


Figure 3.3: A schema of a simulation model composition of the tram crossing

3.2.3 Data Models

Four types of data are available for the model generation in Example 3.1: (i) a CAD drawing of the rail infrastructures, (ii) timetables, (iii) service routes, and (iv) halting time distributions. Table 3.1 summates how they are to be used for the simulation model. The infrastructure data is a basis for model composition and the other three are for model configuration. **Model configuration** refers to

- the setting of the values of a set of invariables or constants, i.e., parameters in a model, and/or
- the setting of the initial values of a set of variables in a model,

where the constants and/or the variables can be in primitive data types or have more advanced data structures. Model configuration can be straightforward when the values to be set are known, and the constants and/or the invariables to be set are locatable in the simulation model. If the values to be set are unknown, we may perform model calibration, which is discussed in § 3.4.3 and § 6.

Figure 3.4 shows the data models of the four data types. The use of data (ii), (iii), and (iv) in the simulation model is therefore rather simple as we have unique identifiers (IDs) to locate and allocate the relevant data.

Comparing Figure 3.2 (p. 41) with Figure 3.3 (p. 42), we can see that the model composition largely resembles the infrastructure layout visualized by the CAD system. In fact, the CAD dataset only contains a list of geometric primitives that describe each object[2] (FLYNN and JAIN 1991). Figure 3.2 shows an example of the list on the right side. The list of entities contains geometric descriptions such as the start and end points of an entity, but the relations between the entities are not represented by data structures. Figure 3.4 (left) shows the infrastructure data model. As expected, it does not directly contain any logical relations. The data model is flat.

Based on a flat data model, we want to construct a simulation model that has a hierarchical compositional structure. This is a major challenge in AMG. Existing data is often poor in content and structure while a simulation model often has rich content and

	Data Type	How to be used in the simulation model
(i)	Infrastructures	Determine the model composition.
(ii)	Timetables	Configure the timetables at each stop and source: one timetable per service line.
(iii)	Service routes	Configure the switch movements at the crossing: one switch position per service line per switch.
(iv)	Halting time distributions	Configure the halting time distributions at each stop: one distribution per service line.

Table 3.1: The data types and their usages for the simulation model for Example 3.1

[2] A CAD object is termed as *entity* in AutoCAD® (www.autocad.com), and as *element* in MicroStation® (www.microstation.com).

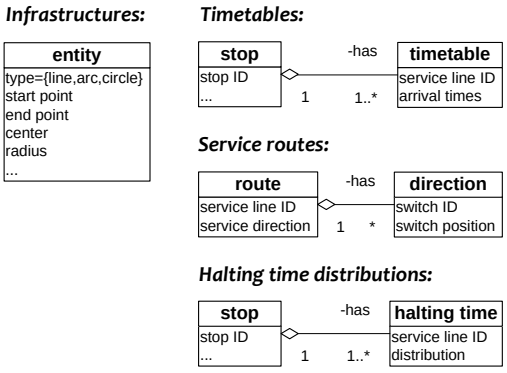


Figure 3.4: Data models for Example 3.1

structure. The content here refers to the portion of information that is related to the system (or model) parts. The structure is related to the relations on the parts.

3.3 Data Quality Issues

In this section, we discuss some issues that one may encounter due to differences of the information contained in data models and simulation models. Data quality issues are often discussed in literature. We focus our discussion on AMG in how well the information in existing data represents the parts and relations of the simulation models under interest.

3.3.1 Background

Some authors make no distinction between *data* (*quality*) and *information* (*quality*) (e.g., WAND and WANG 1996, PIPINO et al. 2002, LOSHIN 2011) while some others see the difference as being crucial (e.g., ACKOFF 1989, LILLRANK 2003, PRICE and SHANKS 2005). The latter argues that data is merely a set of invariant, unprocessed and unstructured symbols recorded in some media (nowadays many of which are digital) whereas information is data ascribed with contextual "meaning attribution"[3].

Information can be inferred from data. The data-to-information transformation process is not viable without *semiotic clarifications*, namely syntactic clarity (i.e., comprehensibility), semantic clarity (i.e., meaning) and pragmatic clarity (i.e., relevance) (CHECKLAND and HOLWELL 1998, ULRICH 2001, ZINS 2007, ROWLEY 2007). In this thesis,

[3] Attributing meaning to data is indeed a human ability; it is done in a context which may well be shared by some but may also be unique to an individual, i.e., the same data may have a different meaning to different people (CHECKLAND and HOLWELL 1998, ULRICH 2001, ZINS 2007, ROWLEY 2007).

we use the term *data* (*quality*) in a broad sense since the use of data is intended to extract information instead of uses purely on a symbolic or syntactic level.

In general, data is deemed as of high quality when it is fit for the intended use (JURAN 1988) and/or when it conforms to its correspondent real-world objects and phenomena (WANG and STRONG 1996, PRICE and SHANKS 2005). In literature, the former view of data quality is known as subjective, user/customer-defined or service-based, focusing on business-world alignment; the latter is known as objective, system-defined or product-based, focusing on real-world alignment. The two views are often seen as competing. PRICE and SHANKS (2005) argue that data quality can be defined and evaluated without sacrificing scope, that is, definition of data quality can and should incorporate both subjective service-based and objective product-based perspectives in one coherent framework. This combined view is also supported by many other authors such as VAN DER PIJL (1994), PIPINO et al. (2002), KNIGHT and BURN (2005) and PETTER et al. (2008). It is the approach we take dealing with data quality issues in AMG.

3.3.2 Data Quality Categories and Criteria

Quality of data is an elusive concept (LILLRANK 2003). It has been investigated and reported extensively in prior research, and a number of data quality frameworks and methodologies has been proposed in literature; see, e.g., surveys from LEE et al. (2002), KNIGHT and BURN (2005), BATINI et al. (2009) and SIDI et al. (2012). Nevertheless, there are many discrepancies in the definitions of data quality criteria (sometimes called dimensions) and in the grouping of criteria into categories, presumably due to the contextual nature of data quality (BATINI et al. 2009). In the following, with respect to the context of AMG we propose data quality categories and criteria selected and adapted from literature.

3.3.2.1 Quality Categories

The work of PRICE and SHANKS (2005) and TEJAY et al. (2006) calls for using *semiotics* as a theoretical basis for defining data quality categories. MORRIS (1938, 1971) introduces the terminology of *syntactics*[4], *semantics*, and *pragmatics* in semiotics[5]. These three dimensions of semiotics are used in other fields often with broader definitions than they originally are in language signs (NÖTH 1995). It is not hard to observe correspondences between semiotics and data (quality) at syntactic, semantic and pragmatic levels. After all, data is a stored *representation* of a represented external object and/or phenomenon as the *referent*, which awaits human or machine *interpretation* and *use*, i.e., the data under interest has (1) form, (2) meaning and (3) application (PRICE and SHANKS 2005).

[4] Syntactics is renamed by some followers as *syntax* according to NÖTH (1995).

[5] MORRIS (1938) identifies three main factors involved in semiotics: "that which acts as a sign, that which the sign refers to, and that effect on some interpreter in virtue of which the thing in question is a sign to that interpreter"; they are called respectively as the *sign vehicle*, the *designatum*, and the *interpretant* (NÖTH 1995). Syntactics studies the relation between a given sign vehicle and other sign vehicles. Semantics studies the relations between sigh vehicles and their designata. And pragmatics studies the relation between sign vehicles and their interpreters (MORRIS 1938, NÖTH 1995).

The three semiotic levels are used to define the corresponding data quality categories (2005):

(A) Syntactic quality category The degree to which stored data conforms to database rules, i.e., data models.

(B) Semantic quality category The degree to which stored data corresponds to represented referents that are relevant to the purposes for which the data is stored.

(C) Pragmatic quality category The degree to which stored data is suitable for a given use.

The syntactic and semantic categories are related to the objective quality view, and the pragmatic category is related to the subjective quality view as discussed in § 3.3.1. Incorporating *both* views (i) provides a comprehensive description of data quality, (ii) facilitates comparison between different quality perspectives, helping check for discrepancies that are likely to signify a quality issue (that may not be immediately obvious from only one view), and (iii) may help in the analysis of the sources of the issues (ibid.).

3.3.2.2 Quality Criteria

For each quality category, we choose, adapt and define data quality criteria that are pertinent[6] to the context of AMG based on theory, literature and pragmatism.

Accuracy Although many data quality studies include accuracy as a key criterion, there is no commonly accepted definition of what it means exactly[7], in particular its difference to quality criteria such as correctness (WAND and WANG 1996, PRICE and SHANKS 2005). We deem data as being accurate when the data values stored in the database are in conformity with the actual or defined values following BALLOU and PAZER (1985) and FOX et al. (1994). Moreover, we distinguish syntactic accuracy from semantic accuracy (SCANNAPIECO et al. 2005, BATINI et al. 2009). **Syntactic accuracy** is the conformity of a data value v to the corresponding definition domain D of the data value (BATINI et al. 2009). **Semantic accuracy** is the conformity of a data value v to its real-world value v' that is considered correct (FOX et al. 1994, REDMAN 1996, BATINI et al. 2009). For example, that v="Yilin" when v'="Chris" is considered syntactically accurate but semantically inaccurate if the definition domain is specified as, e.g., character string. Precision as a data quality criterion does not pertain to semantic accuracy according to this definition, as the conformity can be defined to be an approximation, i.e., $v \approx v'$, to tolerate imprecision.

Completeness WAND and WANG (1996) and BATINI et al. (2009) define completeness as the degree to which a given data collection includes data describing the corresponding set of real-world objects and phenomena. Returning to the definitions of quality

[6] Criteria such as accessibility, believability, privacy and security are important but are not considered relevant for this research.

[7] For example, GELBSTEIN (2003) defines accuracy as the opposite of an error.

categories in § 3.3.2.1, we can see that data completeness can be both semantic and pragmatic. **Semantic completeness** is the degree to which existing values are included in a data collection relevant to the purpose for which the data is stored[8] (REDMAN 1996, BOVEE et al. 2003, PRICE and SHANKS 2005). Consider a data collection of employees' birth dates and driving license numbers. A `null` value signifies incompleteness for a birth date, but not necessarily for a driving license number as not everyone by default has a driving license. On the other hand, the nonoccurrence of `null` values does not necessarily entail semantic completeness. When, e.g., a new employee's data does not appear in the database at all, the data is incomplete although there is no `null` value that signifies this incompleteness. We may call *missing values* of this nature *missing record*. Data completeness is largely related to the purpose of data use. **Pragmatic completeness** is considered in relation with the purpose of data use rather than the (original) purpose for which the data is collected. WANG and STRONG (1996) define this as the degree to which data is of sufficient breadth, depth, and scope for the purpose of data use, i.e., whether there is missing information for a given use of the stored data.

Consistency Intuitively, data is deemed consistent when there is no contradiction or disagreement in the stored data (ibid.). Data consistency issues can be found in syntactic and semantic categories. **Syntactic consistency** refers to uniformity in the (syntactic) representation of data values that have same or similar semantics (PIPINO et al. 2002, LOSHIN 2011). This means that data with same semantics should best share the same underlying syntactic formats and structures. **Semantic consistency** refers to the conformity of (explicit or implicit) semantic rules over a set of data attributes and values (BATINI et al. 2009). Ideally, similar data attributes should share consistent names and meanings (LOSHIN 2011) and inter-related attribute values should not have conflicting or unaccountable meanings. PRICE and SHANKS (2005) further differentiate consistency in key values with that in non-key values[9].

Semantic consistency in key values – let us call it **mapping consistency** – refers to the uniformity in the key values of data representing the same external instance (ibid.). More specifically, when keys assigned with different values indeed intend to map the same external instance, these keys are considered semantically inconsistent although they may be syntactically consistent. Mapping inconsistency often occurs across databases or data repositories. Some authors call it identifiablility or object identification problem (e.g., BATINI and SCANNAPIECO 2006, LOSHIN 2011).

Semantic consistency in non-key values[10] is frequently mentioned in literature, and evidently it often appears in data. Nonetheless, we *exclude* it as a data quality criterion.

[8] Semantic completeness is often related to the `null` values in a database; a `null` value connotes a missing data value (i) that exists but is not known, (ii) that does not exist, or (iii) that is not known whether it exists or not (REDMAN 1996, PRICE and SHANKS 2005, BATINI et al. 2009). Only the first case is seen as being semantic incomplete by our definition.

[9] A key (or mapping) value maps a (non-key) data value (or units) to a represented external (e.g., real world) instance; a non-key value is a representation (of an attribute) of the external instance itself (PRICE and SHANKS 2005).

[10] For example, when person A's marital status is "married" and person A's spouse is person B, there would be a semantic inconsistency if person B's marital status is "single".

Inconsistency often occurs in real world objects and phenomena. It does not necessarily indicate erroneous data values and is therefore not a valid criterion for data quality. For those data values that do require semantic consistency[11], the data quality is covered by the semantic accuracy criterion, as these values can not be accurate when they are not consistent.

Referring back to the accuracy criteria, they concern both key values and non-key values. Semantic accuracy in non-key values implies semantic accuracy in mapping (i.e., the associated key-values), not vice versa. More specifically, when a non-key value is semantically accurate, it has a meaningful and unambiguous mapping[12].

Timeliness There are many definitions of timeliness of data. It may refer to the time expectation for accessibility of data (e.g., LOSHIN 2011), the delay between a change of a real world state and the resulting modification of the information system state (e.g., WAND and WANG 1996), or how up-to-date the data is with respect to the task it is used for (e.g., WANG and STRONG 1996, PIPINO et al. 2002), etc. Some authors (e.g., FOX et al. 1994, CATARCI and SCANNAPIECO 2002, BOVEE et al. 2003, BATINI and SCANNAPIECO 2006, BATINI et al. 2009) characterize timeliness with sub-criteria such as *currency* (i.e., how recent is the data, or how promptly the data is updated) and *volatility* (i.e., how long the data remains valid, or how frequently the data varies in time). Timeliness in this thesis is in the pragmatic category and it refers to the extent to which data is within a valid time frame with respect to the purpose of data use (PRICE and SHANKS 2005).

Presentation Suitability This criterion is in the pragmatic quality category and it refers to the degree to which the data format, unit, precision and type-sufficiency are appropriate for the purpose of data use (PRICE and SHANKS 2005, MCGILVRAY 2008). Data precision[13] is defined as the degree to which each data value expresses sufficient detail that is appropriate for the purpose of data use (PIPINO et al. 2002, PRICE and SHANKS 2005); type-sufficiency refers to the degree to which the data includes all the types of information useful for the purpose of data use (PRICE and SHANKS 2005).

To provide an overview, Table 3.2 numerates the proposed data quality categories and criteria. Table 3.3 summarizes the definitions of the eight criteria ordered to the three categories.

[11] For example, an under-age child can not be "married", neither does the child have a driving license.

[12] A non-key value is meaningfully mapped when it refers to at least one specific external instance; it is unambiguously mapped when it refers to at most one specific external instance (PRICE and SHANKS 2005).

[13] When precision is related to measurement systems, it is the degree to which repeated measurements under unchanged conditions show the same results (TAYLOR 1999). As for data quality, we support the view that data precision should be considered with respect to data use (FOX et al. 1994, LEVITIN and REDMAN 1995, PRICE and SHANKS 2005). Data precision without context is often meaningless. After all, no real measurement is infinitely precise.

	Accuracy	Completeness	Consistency	Timeliness	Presentation Suitability
A. Syntactics	#1.	-	#2.	-	-
B. Semantics	#3.	#4.	#5. (mapping)	-	-
C. Pragmatics	-	#6.	-	#7.	#8.

Table 3.2: Proposed data quality categories and criteria

3.3.3 Discussion On Data Quality Issues and Measures

In the context of AMG with respect to component-based modeling (i.e., the pragmatic use of data in this thesis), we assess data quality according to *how well the data provides information for model component identification, composition and configuration.* The pragmatic use of data influences the *perceptions of syntactics and semantic criteria* is argued by PRICE and SHANKS (2005) whose work includes this as a quality criterion in the pragmatic category. We support this view but do not explicitly include this criterion. We assume that the data use for AMG is a given goal, and data users shall consider the syntactics and semantic criteria with the application domain.

Detecting and solving data quality issues need *domain understanding* and *data understanding.* The purpose of defining the data quality categories and criteria is to help detect and solve data quality issues for AMG. In literature, methodologies for data quality assessment and improvement are discussed in the general context of information systems and data management (see, e.g., the survey by BATINI et al. 2009). In this research, as stated earlier, data quality is assessed in relation with how well it provides information about model parts and relations for AMG.

To give examples, we refer back to Example 3.1 in § 3.2 (pp. 40∼44) where data from four sources is available for generating the tram crossing model. The model schema (Figure 3.3) and the data models (Figure 3.4) show that the parts and relations in the former is richer than those in the latter. In Table 3.4, we list the major data issues appearing in Example 3.1.

Data issues in the syntactic category are often straightforward. Syntactic accuracy (#1.) is related to the lawfulness, rather than the correctness, of data values (WAND and WANG 1996). In many information systems, it can be automatically checked by *comparison functions* (BATINI and SCANNAPIECO 2006). Syntactic consistency (#2.) is particularly relevant when data is being sourced from multiple information systems (SHANKS and CORBITT 1999). Syntactic inconsistency can be typically solved through data type and format conversion.

In order to measure semantic accuracy (#3.) of a data value v, (i) the corresponding true value v' has to be known, or (ii) it should be possible, considering additional knowledge, to deduce whether v is or is not v' (BATINI and SCANNAPIECO 2006). The first option is a non-option in a computational sense. If the "true value" is or can be known digitally, that value should be used instead of v. Hence, semantic accuracy is

Category	#	Criterion	Definition	Reference	Example/Explanation
A. Syntactics	1.	Syntactic accuracy	The conformity of a data value v to the corresponding definition domain D of the data value.	SCANNAPIECO et al. (2005) and BATINI et al. (2009)	Accurate: v="Yilin", v'="Chris" Inaccurate: v="Yilin", v'=2012 for D=`varchar`
	2.	Syntactic consistency	The uniformity in the syntactic representation of data values that have the same or similar semantics.	PIPINO et al. (2002) and LOSHIN (2011)	Inconsistent: in table-1 employee.id=1234 while in table-2 enrollment.id="1234".
B. Semantics	3.	Semantic accuracy	The conformity of a data value v to its real-world value v' that is considered correct.	FOX et al. (1994), REDMAN (1996) and BATINI et al. (2009)	Accurate: v=10.1, v'=10 Inaccurate: v=11, v'=10 for $\lvert v-v'\rvert \leq 0.1$
	4.	Semantic completeness	The degree to which existing values are included in data relevant to the purpose for which the data is stored.	REDMAN (1996), BOVEE et al. (2003) and PRICE and SHANKS (2005)	Incomplete: there are missing values in the data.
	5.	Mapping consistency	The uniformity in the key values of data representing the same external instance.	PRICE and SHANKS (2005)	Inconsistent: keys assigned with different values intend to map the same external instance.
C. Pragmatics	6.	Pragmatic completeness	The degree to which data is of sufficient breadth, depth and scope for the purpose of data use.	WANG and STRONG (1996)	Incomplete: there is missing information for a given use of the stored data.
	7.	Timeliness	The extent to which data is within a valid time frame with respect to the purpose of data use.	WANG and STRONG (1996) and PRICE and SHANKS (2005)	University course schedules are valid for a given time frame (e.g., a particular semester).
	8.	Presentation suitability	The degree to which the data format, unit, precision and type-sufficiency are appropriate for the purpose of data use.	PRICE and SHANKS (2005) and MCGILVRAY (2008)	Data format, unit, precision and type-sufficiency are sub-criteria.
		Precision	The degree to which each data value expresses sufficient detail that is appropriate for the purpose of data use.	PIPINO et al. (2002) and PRICE and SHANKS (2005)	The appropriateness of image resolution is dependent on the use of images.
		Type-sufficiency	The degree to which data includes all of the types of information useful for the purpose of data use.	PRICE and SHANKS (2005)	

Table 3.3: Definitions of data quality criteria

only computationally measurable and solvable with sufficient knowledge to reason the deduction. The two issues of semantic accuracy listed in Table 3.4, e.g., can be measured and corrected (if inaccurate) with the support of domain knowledge.

Data completeness (#4. and #6.) issues can be found in both semantic and pragmatic categories. In either case, when data is *truly* incomplete[14], we can only complete the data by acquisition of the missing parts. In dealing with the pragmatic incompleteness in Table 3.4, e.g., the three missing vehicle generation points are added into the existing infrastructure data manually. Improving semantic completeness could increase the chance of pragmatic completeness in potential data uses. Nonetheless, semantic incompleteness does not necessarily signify pragmatic incompleteness.

Mapping consistency (#5.) issues typically occur among data across different sources. Sometimes mapping consistency is broken because of erroneous schema changes (VELEGRAKIS et al. 2004). When key values intended to map to the same external instance are inconsistent, a mapping table can be provided to clarify the relations among these keys.

Time can affect the validity of data. Given a time frame of data validity, timeliness (#7.) is easy to measure if data has metadata or dedicated fields (i.e., timestamped) to indicate its time attributes, e.g., when the data is collected or updated and how long it is valid. In Example 3.1, there is no timeliness issues in the data; the timetable data, for example, is timestamped, so the users only need to pay attention to querying the data

#	Criteria	Data issues
1.	Syntactic accuracy	–
2.	Syntactic consistency	Data from different sources do not share coherent syntactic rules.
3.	Semantic accuracy	Some track entities (in the infrastructure data) have directions that do not correspond to the direction of traffic flow. Some entities (in the infrastructure data) supposed to be connected do not have the same coordinate values.
4.	Semantic completeness	–
5.	Mapping consistency	Some stop IDs in the infrastructure data are different from the ones in the timetable data.
6.	Pragmatic completeness	The vehicle generation points are not contained in the infrastructure data.
7.	Timeliness	– (*The data is timestamped.*)
8.	Presentation suitability	The relations between entities (in the infrastructure data) are not explicitely represented. There are no entities (in the infrastructure data) that correspond to model components 7~11 (p. 41).

Table 3.4: Data issues in Example 3.1 in § 3.2

[14] Meaning that (i) the data values or records are unknown but do exist, (ii) they are not contained by other accessible data sources, and (iii) they are not deducible from known data values or records.

with the correct time expressions.

Presentation suitability (#8.), particularly type-sufficiency, poses many data issues in AMG. As discussed in, e.g., § 1.1, § 3.1 and § 3.2, existing data often does not contain the right content and structure of information required for model generation, i.e., the parts and relations in the simulation model is richer than those represented in the data. We do not deem this problem as being semantic incomplete, because the missing information can be deduced from the existing data with sufficient domain knowledge. The data is not (truly) incomplete but the information directly contained is not of the right type. When the domain knowledge and reasoning for deduction can be formalized, we are able to obtain the right type of information automatically from the data. This can be achieved through model transformation discussed in § 5.

Remarks To conclude, with regard to the information of model structure and parameterization, the data that is provided to the AMG as input should be assessed according to how well the data provides information for model component identification, composition and configuration. The requirements for the data (assume that the date has syntactic accuracy and timeliness) should have:

(1) semantic and pragmatic completeness, and

(2) syntactic and mapping consistency, or conversion rules or mapping tables or alike that can solve the inconsistency in the data.

Transformation rules for the AMG can be defined when modelers have sufficient domain knowledge and deductive reasoning to solve issues related to semantic accuracy and presentation suitability.

3.4 From Data to Simulation Model

Domain Model Component Library The goal of AMG in this research is to generate simulation models according to existing data. As a prerequisite for an AMG process, model components need to be developed in the first place. As discussed in § 3.1, we need a model component library (Construct 2) that:

- conforms to the domain meta-model (Construct 1),
- implements the concept of hierarchical component-based modeling (§ 2.3), and
- uses an appropriate underlying modeling formalism (§ 2.4).

As we will show in § 4, the specifications of the domain meta-model are defined as a part of the domain model component library we developed for this research.

An AMG Process Starting from the existing data to a simulation model as a final outcome, we need to design an AMG process that:

- executes a model transformation definition (Construct 4) specified on the data models (Construct 3) and the domain meta-model (Construct 1),

- generates (or instantiates) a simulation model using the predefined model components (Construct 2), and
- performs model calibration (Construct 5) after the initial generation of the simulation model.

Accordingly, we propose to divide an AMG process into three steps: *model transformation, model instantiation* and *model calibration*, as shown by Figure 3.5.

Some Terms in Model Transformation SYRIANI (2011) considers a model transformation as the automatic manipulation of a model with a specific intention. According to KLEPPE et al. (2003) and MENS and VAN GORP (2006), a **model transformation** is the automatic generation of a **target model** from a **source model** according to a transformation definition; a **transformation definition** is a set of transformation rules (and units) that together describe how a model in the source language can be transformed into a model in the target language. A **transformation rule** describes a small unit used to specify a transformation (CZARNECKI and HELSEN 2006). A model transformation may have multiple source models and multiple target models (MENS and VAN GORP 2006). In AMG, it is possible and likely to have multiple source models, which are data from different sources, but the target model is in principle a single simulation model (for each referent of interest).

Basic Elements in Model Transformation The model transformation rules are defined on meta-models whereas the model transformations are performed on concrete models according to the predefined rules. Following CZARNECKI and HELSEN (2006), the basic elements involved in model transformation are shown in Figure 3.6:

- a source (or input) model that conforms to a source meta-model;
- a target (or output) model that conforms to a target meta-model;

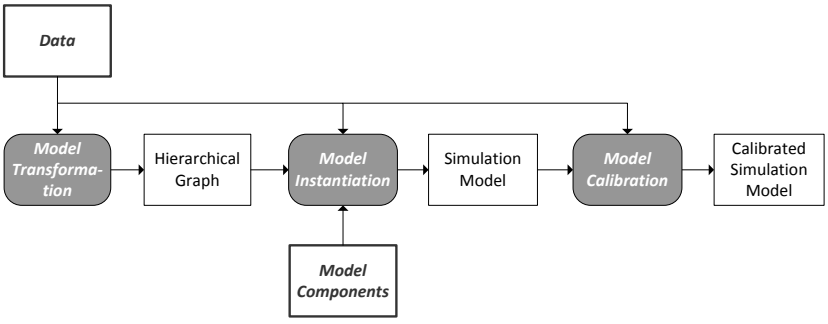


Figure 3.5: Proposed steps in an AMG process

- a transformation definition that is specified referring to the source and target meta-models; and
- a transformation engine that reads the source model, executes the transformation definition on it, and writes the target mode.

In our research, the (first) source meta-model is the data model (Construct 3), and the (last) target meta-model is the domain meta-model of interest (Construct 1), on which the transformation rules (Construct 4) are defined. The source and target models are respectively the existing data of a referent and the to be generated simulation model of the referent. Both conform to their meta-models accordingly as the meta-models are their type models (§ 2.2.1). The transformation engine reads the data, executes the transformation rules on the data, and then generates the simulation model using the predefined model components (Construct 2).

Vertical and Horizontal Transformations Model transformations can be characterized as vertical, horizontal or oblique, as illustrated in Figure 3.7 (CZARNECKI and EISENECKER 2000). In a **vertical transformation**, the source and target models reside at different abstraction levels; the model often has content refinement; a **horizontal transformation** is a transformation in which the models reside at the same abstraction level; the structure of the model is redefined (ibid., MENS and VAN GORP 2006). Formal refinement and code generation are typical examples of vertical transformations (ibid.); the transformation of a UML data model to the corresponding relational database schema is horizontal (CZARNECKI and HELSEN 2006). Refactoring (FOWLER 1999) is widely known for its horizontal transformations (MENS and VAN GORP 2006), though its transformations can be vertical as well (CORNÉLIO 2004). An **oblique transformation** combines vertical and horizontal transformations.

Transformations in AMG are often oblique. As stated in § 1.1, existing data frequently does not contain the right type of information for AMG and needs to be transformed both in content and structure. In an AMG process, the model transformation step is mainly concerned with horizontal transformation by which a model composite structure should be generated. The model instantiation step is mainly concerned with vertical transformation by which a simulation model is generated according to the

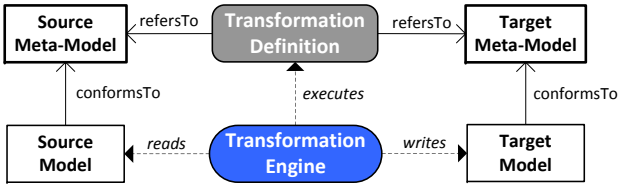


Figure 3.6: Basic elements in model transformation (ibid.)

model composite structure.

3.4.1 Model Transformation

The step of model transformation concerns Construct 1, 3 and 4: a *meta-model* of the domain of interest, *data models* of the data of interest, and a *model transformation definition* specified on the former two. In this first step, based on relevant data considering data quality issues discussed in § 3.3, we need to construct a *hierarchical graph* that is a homomorphic representation (see § A.1) of the simulation model to be generated. Since we use hierarchical component-based modeling and the DEVS formalism, the model composition can be conveniently represented by a directed hierarchical graph. The theoretical works in graph transformations are in this case applicable to model transformations.

Intermediate Transformation Steps As just stated, transformations in AMG are typically oblique because existing data often needs to be transformed both in content and structure for AMG. The model transformation can be of considerable heavy-duty when the information gap between the data model and the domain meta-model is large (MENS and VAN GORP 2006). The transformation of some parts and relations may be dependent on the transformation of some other parts and relations. Consequently, the model transformation may itself need intermediate steps. The application of intermediate transformation steps is sometimes called *transformation composition* or *transformation chain* (ASZTALOS et al. 2011). The steps should incrementally transform the data into a hierarchical graph that is homomorphic or isomorphic to the simulation model composition. Each intermediate step constructs an intermediate structure which require its own meta-model (CZARNECKI and HELSEN 2006).

Meta-Models and Graph Patterns In § 2.2, the roles and types of meta-models were explained. In § 3.1, we explained the reasons why meta-models are needed in this research. In § 3.4, we showed that meta-models are basic elements in model transform-

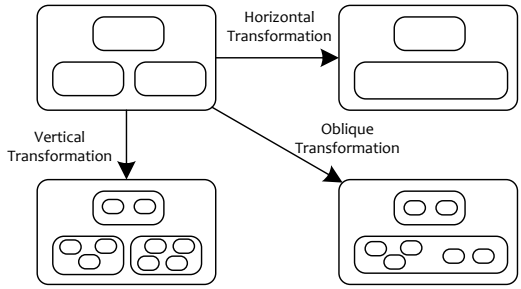


Figure 3.7: Vertical and horizontal transformations (CZARNECKI and EISENECKER 2000)

ation. When model composition is represented by graphs, model transformations can be treated as graph transformations. In graph transformation, *graph pattern matching* precedes *graph rewriting*, both of which have wide applications (KAHL 2002, GALLAGHER 2006). Graph patterns are meta-models too. They can be assembled into more complex patterns by *pattern composition* (BALOGH and VARRÓ 2006, ASZTALOS et al. 2011) which is sometimes termed *pattern call* (VARRÓ and BALOGH 2007, VARRÓ et al. 2008). Graph pattern composition suits well with the component-based modeling concept and is therefore used in this research. The composition of graph pattern matching in this thesis is performed in a stepwise bottom-up manner.

Transformation Rules and Rule Application Control Transformation rules need to be carefully designed to identify and correct or tolerate data quality issues (if any) discussed in § 3.3. Although data quality issues also appear in model instantiation and calibration, they often post more problems in the model transformation step, as the transformation in this step is mainly horizontal and involves structural changes (CZARNECKI and EISENECKER 2000).

A transformation rule should specify the elements or a pattern of elements in the source model (and target model), and expresses logics (e.g., relations, constraints and computations) on the elements (CZARNECKI and HELSEN 2006). At the same time, a rule may also have application conditions (which must be true in order for the rule to be executed) and control parameters, etc., to make it more flexible (ibid.). A *rule application control* specifies *where* the transformation rules are executed on the source model and *in what order* the rules are applied (SYRIANI 2011). The former is called *location determination* and the latter called *rule scheduling* (CZARNECKI and HELSEN 2006). Depending on the goal of the transformation, rule application control can be deterministic or non-deterministic (ibid.). There may be multiple locations, i.e., more than one match, for applying a rule. Rule scheduling mechanisms can determine the application order of rules implicitly, e.g., through designing the patterns and logic of the rules to ensure certain execution orders, or explicitly, e.g., by specifying conditions for rule selection or iteration (ibid., SYRIANI 2011).

We choose to design transformations that are deterministic in the AMG method. The rules are defined in a deterministic order (partially ordered). Each rule is executed sequentially for matching and rewriting in the source graph. Therefore, no rules would compete for the same source location and the transformation shall produce the same hierarchical graph given the same input data.

3.4.2 Model Instantiation

The second step starts with the *hierarchical graph* produced by the model transformation step and a set of predefined *model components* (Construct 2). The hierarchical graph serves as the blueprint that guides the model composition for model instantiation, where the model components are the building blocks.

A simulation model is constructed according to the hierarchical graph according

to which model components are instantiated and coupled. Each node (or edge, depending on whether the graph is node-oriented or edge-oriented) in the hierarchical graph should have a corresponding atomic or coupled model component that can be identified. Relevant data is used to configure the model component instances. Model configuration (i.e., setting the initial values of variables and the values of parameters, § 3.2.3) takes place at the elementary level of the model components (i.e., in the atomic models). The parameters whose values are not provided by the data are configured by predefined default values.

3.4.3 Model Calibration

To complement the AMG method, the generated simulation model can be calibrated (Construct 5) when empirical data from the system is available. This can be performed as a third step of the AMG process. In this research, model calibration is an experimental process by which some model parameters whose values are unknown are adjusted in each simulation experiment to match simulation output (data) with relevant systems observation or measurement given prespecified acceptance criteria. Comparing simulation output with relevant systems measurement (i.e., empirical data, § 2.1.2), if the latter exists, is a model validation method (BALCI 1998). Together with this validation method, the parameters configured by default values (in the previous step) can be calibrated to obtain a reasonable agreement between the simulated system and the actual system in terms of their behaviors (MUSSELMAN 1998).

The AMG method proposed by this research is presented with three parts: domain simulation library (§ 4), model generation (§ 5), and model calibration (§ 6). In § 4, we discuss the theory and the research findings related to developing a domain simulation library for AMG. In § 5, model transformation and instantiation (i.e, the transformation rules) are discussed based on graph transformation theory. In § 6, a model calibration procedure is presented.

3

4

Domain Simulation Library

AFTER THE DISCUSSION in § 2 and § 3, it should be clear that a model component library of the application domain needs to be developed in the first place so as to provide reusable building blocks for the AMG process proposed. In order to obtain functional and usable simulation models, we also need components that allow for the use of the model components. Examples are those for model execution (i.e., a simulator), output collection and model/simulation visualization. All these components together constitute a simulation library.

In software engineering, the reuse of components has been an important development goal (SOMMERVILLE 1996, BRAUDE and BERNSTEIN 2010). The software community learned that frameworks such as Java APIs lend themselves to highly successful reuse (BRAUDE and BERNSTEIN 2010). A software *framework*, sometimes called a *library*, is a collection of software artifacts usable by several applications[1] (ibid.). The motivation of developing a simulation library is congenial, despite that a simulation library is specialized in or tailored to particular uses of M&S.

In this chapter, we present the theory, our experience and research findings related to developing a simulation library that is fit for use for AMG. The cases in develop-

[1] In other words, libraries are collections (or packages) of reusable components designed to implement a general solution to a general problem (BRAUDE and BERNSTEIN 2010). A library typically begins to emerge by the time a development organization develops its second to fourth application (ibid.).

ing a library (called LIBROS) for the rail transportation domain are used as examples through the chapter. LIBROS started forming after a number of M&S studies in the light-rail domain since 2008, e.g., network design studies of *RegioTram* (in *Groningen*) and *RijnGouwelijn* (in South Holland) in The Netherlands. It is then progressively developed and extended along with more studies for this research. In § 4.1, we discuss the application context, challenges and functional elements of the LIBROS simulation library. Thereafter, the conceptual and refined design of the library is presented through the rest of this chapter. The focus of this chapter is § 4.2 and § 4.3 which discuss the model components.

4.1 Towards Developing A Rail Simulation Library

M&S of transport systems is recognized by many transport organizations as an effective decision support instrument (ORTÚZAR and WILLUMSEN 2001). In order to successfully support detailed design and operation, a *microscopic* rail network model is often deemed not only suitable but also mandatory (HANSEN and PACHL 2008). A rail model should have *sufficient detail* and *accuracy* to represent the complex and often large-scale system, yet still be *computationally efficient*. Apart from these criteria, considering the life span of railway systems, the number of studies demanded and the time and cost induced, *reusability* and *extensibility* of simulation models is often an issue that should be addressed during model design.

4.1.1 Application Context and Challenges

Briefly introduced in § 1.1.2, HTM is a public transport organization that provides light-rail transport (and other public transport) services in *Haaglanden* region. In close co-operation with HTM, a Java *Library for Rail Operation Simulation* (LIBROS) is developed within the context of this research. The organization's interest in this simulation library is manifold:

Modeling Driving-On-Sight Many rail simulation tools exist[2], yet very few[3] support modeling of urban railway such as tramway or light-rail. Heavy-rail vehicles generally operate in *block systems* that provide strict safe spacing control by signaling[4], whereas light-rail vehicles also *drive-on-sight* (OVERTON 1989). In the latter case, modeling the

[2] For example, models of stations or terminals (CAREY and LOCKWOOD 1995, RIZZOLI et al. 2002, CAREY and CARVILLE 2002, 2003), and train network simulators such as Simon/TTS (WAHLBORG 1996), TOPSim (SANDBLAD et al. 2000), RailSys (BENDFELDT et al. 2000), UX-SIMU (KAAS 2000), VirtuOS (KAVICKA and KLIMA 2000), SIMONE (MIDDELKOOP and BOUWMAN 2001), Multi-train simulator (HO et al. 2002), OpenTrack (NASH and HUERLIMANN 2004), SimMETRO (KOUTSOPOULOS and WANG 2007).

[3] To the author's knowledge, one of the few is RailSys (RUDOLPH 2000).

[4] Non signal-controlled modern railway operation (i.e., the communication between the dispatcher and the train crew is made via telephone or radio) is quite rare in Europe, and can only be found on branch lines with a very low density of traffic (PACHL 2002).

interactive movement of vehicles is necessary as the vehicles do not only drive according to signals (RUDOLPH 2000). HTM operates urban railway in *Haaglanden* region and provides rail related consultancy nationally and internationally. Given the large number of urban areas with light-rail systems and the increasing acceptance of M&S as a method of inquiry (SOL 1982, KEEN and SOL 2008), there is a growing need for tools that allow for light-rail modeling.

Microscopic Modeling of Rail Operation Many rail simulation tools are specialized in a certain aspect of the railway, e.g., to study stations (CAREY and CARVILLE 2002, 2003), to assess timetables (MIDDELKOOP and BOUWMAN 2001), or to analyze rail physics and energy consumption (HO et al. 2002). Some models have high abstraction levels (VROMANS et al. 2006), i.e., they are macroscopic, which can be applied for high-level tasks like vehicle scheduling or planning for traffic demand and allocation (HANSEN and PACHL 2008). To the contrary, running time calculation, timetable assessment and rail operational simulation requires detailed infrastructure models (ibid.). This is the type of model required by HTM. The model should allow for the overall analysis of the design and operation considering the impact of factors such as infrastructure layout, signaling systems, timetables and driving behaviors and travel time. Rail simulation models can be of large-scale. The simulation of large-scale microscopic models is often computationally intensive. In this regard, the choice of modeling formalism and the design of models can contribute to the computational performance.

Reusability and Extensibility The components (particularly the model components) in the library are meant to have long life cycles[5]. As stated in § 1.1.2 and § 1.1.3, HTM often uses models which take long to develop. For a certain domain, simulation models can be designed such that a new model may reuse some of the previous modeling solutions. In this research, reuse is focused on the component level[6]. Closely related to reusability, extensibility[7] is required by long-life-cycle models for changes and expansions of functionality (CRNKOVIC et al. 2011). Each system in the domain is similar yet different from one another, and each system itself may and is likely to have changes during its life cycle.

[5] ULGAN and GUNAL (1998) state that a simulation model may have a short or long life cycle based on the use through the life of the system of interest. Short-life-cycle models are for a single decision making purpose; once the decision is made, the model is discarded. Long-life-cycle models are used at many points in time during the life of the system and are maintained and re-validated as conditions change in the system. Such models are generally built for the purpose of (i) training, (ii) reuse at the launch phase, and (iii) reuse at the fully operational phase for changes in design and operation of the system (ibid.).

[6] ROBINSON et al. (2004) recognize a spectrum of different reuses, namely code scavenging, function reuse, component reuse and full model reuse, from a low abstraction level to a high abstraction level. Reuse can also be performed on requirement specifications, tests, among many others (*e.g.* LUBARS 1988) which are not discussed here.

[7] Instead of sufficiency or completeness which is usually an elusive goal (BRAUDE and BERNSTEIN 2010).

4.1.2 Basic Functionality and Elements

Modern simulation software may have rich functionality[8] (BANKS 1998, ROBINSON 2004, PIDD and CARVALHO 2006, LAW 2007, BANKS et al. 2010). There is, however, a set of *basic functionality* that should be provided, i.e., the ability to construct, run and report simulation models.

Functionality For Model Specification It refers to the ways or supports for constructing simulation models, or simply, for modeling. A simulation study needs at least a model on which the simulation experiments are carried out. Models can be specified using simulation languages, parametric models, model components, etc., possibly through graphical user interfaces (ÖREN 1981, PIDD and CARVALHO 2006, BANKS et al. 2010). In our case, as we are interested in hierarchical component-based modeling, we need *model components* that specify the elementary level of model building blocks and the possible *composition* of components.

Functionality For Model Experimentation It refers to the supports for model experimentation which includes (1) model execution and (2) specification of experiments (ÖREN 1981). The former is achieved through an adequate simulation engine or a *simulator*; the latter is defined by *experimental frames* following ZEIGLER et al. (2000).

ZEIGLER et al. (ibid.) identify four basic entities for their M&S framework: *source system*, *model*, *simulator*[9] and *experimental frame*, three of which are discussed in § 2.1. Figure 4.1 shows these entities and their relations. Recall that an experimental frame is a specification of the conditions under which a system is observed or experimented

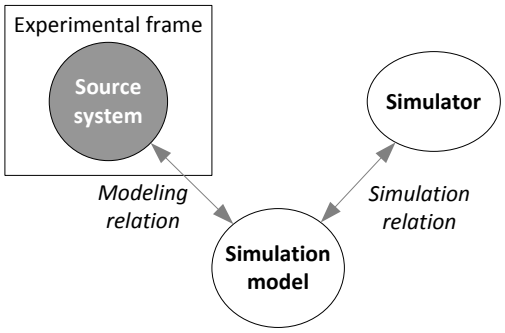


Figure 4.1: Basic elements in M&S and their relations (ibid.)

[8] For example, it may be web-based, support online simulation, or can be integrated into larger environments.

[9] A **simulator** is any computation system capable of executing a simulation model to generate the model's behavior (ZEIGLER et al. 2000).

with[10] (ibid.). Validity refers to the *modeling relation* between a model, a system and an experimental frame. When considered valid, a model can be used in place of its (original) system for the intended purpose of experimentation (§ 2.1.1). The *simulation relation* is a relation between a simulator and a model. A simulator should correctly execute a given class of models according to their specifications, i.e., it faithfully generates a model's output trajectory given its initial state, input trajectory and specification (ibid.).

Functionality For Model/Simulation Presentation It refers to the ways or supports for presenting the models and simulation results. Virtually all modern simulation software provides functionality for *visualization* and *statistics* to support the presentation (and communication) of simulation output to model users as well as to support the verification and validation of simulation models (BANKS 1998, CHIN 2007, LAW 2007, BANKS et al. 2010, FUMAROLA 2011). The use of M&S entails the collection of output data during simulation runs. System performance is typically measured by *Key Performance Indicators* (KPIs) (FUMAROLA 2011) that are aggregated from model *output variables* whose values are computed during simulation runs (ZEIGLER et al. 2000). It is often convenient[11] to record (or log) output values, e.g., into a database. For this purpose, components related to *data output* are needed.

Basic Functional Elements Based on the foregoing discussion, a set of elements are proposed to be included or whose specification should be supported by a simulation library:

1. model components (and composition),
2. simulator,
3. experimental frame,
4. data output components,
5. simulation visualization components, and
6. statistics components.

A domain simulation library should provide a set of model components with which usable models can be composed to simulate a given class of potentially infinite number of systems in the domain. In § 4.2 and § 4.3, we first discuss the modeling concept and then the model components (#1) in LIBROS. As for the simulator (#2) and the experimental frame (#3), we used the previous works of JACOBS (2005) and SECK and VERBRAECK (2009)[12] which will not be discussed in length in this thesis. The components for model/simulation presentation, i.e., model output, visualization and statistics (#4, 5, 6) in LIBROS are briefly discussed in § 4.5.

[10] The running of a simulation experiment is the process of running the model so as to observe and analyze the result to obtain the desired answers (SHANNON 1975).

[11] This can (i) decouple simulation (output) with operations on the output, (ii) reduce the demand for memory space during simulation runs, and (iii) allow users to use the output data at a later point in time.

[12] For completeness, § A.2 shortly presents the simulator used by this research, i.e., DSOL and ESDEVS.

Loose Coupling and Composability A number of literature discusses issues such as qualities of M&S products (e.g., BALCI 2004), criteria for good models (e.g., SHANNON 1975), properties of component software (e.g., SZYPERSKI 2011), or principles of software design in general (e.g., SOMMERVILLE 1996, BRAUDE and BERNSTEIN 2010). Among those, loose coupling and composability are particularly important for hierarchical component-based modeling with respect to AMG.

Out of general software design principle (SOMMERVILLE 1996, BRAUDE and BERNSTEIN 2010), the functional elements in the library should be loosely coupled to one another. The coupling relations refer to the relations between the model components and the other elements, and also the relations between model components themselves. Coupling describes the degree of interconnectedness between components in a design (SOMMERVILLE 1996). The tighter the coupling the harder it is to understand and change the system (BRAUDE and BERNSTEIN 2010). In FUMAROLA (2011), the author argues the importance of loose coupling between simulation models and components that deal with model presentation, e.g., those for animation and statistics. High coupling disadvantages flexibility and clarity (ibid.). When models contain too much information other than model specifications, complicated models become even harder to understand (ibid.). Similar arguments are made by FOWLER (1999) and BRAUDE and BERNSTEIN (2010) who refer to this as *separate domain from representation*.

Separate modeling from simulation is often discussed in M&S literature (e.g., BANKS 1998, ZEIGLER et al. 2000, FUJIMOTO 2000, WAINER 2009). Since developing a simulator is not within the scope of this thesis (see § A.2) we do not present related issues. The only point we shall mention is that a simulator should support the simulation of the chosen modeling formalism(s) of model components in a loosely coupled manner.

Components should support composition (CRNKOVIC et al. 2011). This claim seems redundant, but it is acknowledged by many authors that model composability is difficult to apply (YILMAZ 2004, SZABO and TEO 2007, TOLK et al. 2010). In § 2.3, composability is briefly discussed, so is modularity. Modularity is a prerequisite for component technology; many modularity criteria date back to PARNAS 1972 (SZYPERSKI 2011). Unfortunately, the vast majority of software solutions today are not even modular (ibid.). In software design, the principle for modularization is coined by DIJKSTRA (1982) as *separation of concerns*:

> This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

To modularize effectively, one tries to maximize cohesion and minimize coupling (PARNAS 1972, SOMMERVILLE 1996, BRAUDE and BERNSTEIN 2010). However, besides modularity, composability also requires the adoption of principles of independence and controlled explicit dependencies (SZYPERSKI 2011). In this regard, the DEVS formalism (§ 2.4.3) provides a strong support for hierarchical composition with coupling constrains.

4.2 Systems Modeling

Given a modeling problem, modelers start from a problem situation, move through requirements, and work towards a definition of *what is going to be modeled and how* (ROBINSON 2010). Many challenges in modeling[13] are in the process of *simplification* (and *abstraction*)[14]. Simplification entails the stripping away of unimportant details, or the assumptions of simpler relations (SHANNON 1975). Abstraction comprises or concentrates in itself the essential qualities or behavior of a thing but not necessarily in the same form or manner as in the original (ibid.). Successful modeling can be seen as valid simplification (and abstraction) (ZEIGLER et al. 2000), taking into consideration other relevant factors such as cost, performance, reusability and extensibility.

Model Components and Compositions Model component specification can be divided into two related parts:

1. the specification of the (behavior of) model components at the elementary compositional level, and
2. the definition of the permissible (structure of) model compositions at the non-elementary compositional levels.

The former provides the basic units in constructing hierarchical component-based models which subsequently generate the overall model behavior. In DEVS, model behaviors at the elementary level are specified in atomic models by means of transition functions, time advance functions, etc.

The behavior of a composed model (component) is determined by the collective (interactive) behavior of its sub-components. Not only the behavior of the constituent components but also the alterable composite structure alter the behavior of the composed model. In DEVS, compositions are specified in coupled models by means of couplings. Following the concept of closure under coupling in DEVS (ibid.), the resultant of coupling is also a DEVS model, which can be again used for composition and so recursively.

"The definition of the permissible model compositions" advocates abstract definitions of model composites in order to allow for certain flexibility of composition under predefined constraints. In this sense, the abstract definitions of model compositions are not classic DEVS coupled model specifications. They are defined by patterns of DEVS coupled models through meta-models, and are accompanied by necessary functions that can create (classic) DEVS coupled models with permissible compositions (§ 4.3.5 and § 5.3).

[13] Some M&S literature states that modeling is deemed more of art than science (e.g., SHANNON 1975).

[14] Some literature prefers not to differentiate simplification from abstraction. For example, in ZEIGLER et al. (2000): abstraction is a general process and includes various simplification approaches; the word "abstraction" is not mentioned in ROBINSON (2004).

Systems Decomposition In § 2.1.1.2 and § 2.3, we had some discussion about systems decomposition and related concepts. Modelers typically view systems as interacting parts and relations using the golden rule of *separation of concerns* (DIJKSTRA 1982). Even if not perfectly possible, it is yet the only available technique for effective ordering of one's thoughts (ibid.). Although supported by the near-decomposability principle of SIMON (1996), some decisions for decomposition are not easy to make, especially when the systems are of natural kinds where there are non or indistinct sub-system boundaries. A typical example is fluids, viz., liquids and gases. HOFMANN (2004) discusses different terrain representations, e.g., decomposition with quadratic or hexagonal cells, and ways to represent visibility, soil and surface, etc. The author states that none is superior to all the others in every aspect (ibid.). In such cases, it may be beneficial to consider whether the resulting design is compatible to or interoperable[15] with other simulation systems. After all, modeling choices should be strongly problem solving oriented, the problem being defined for the purpose of model use or higher level goals such as cohesion, low coupling or composability. Here is how BREEDVELD (2009) comments on modeling choices.

> A key aspect of the physical world around us is that "nature knows no domains". In other words, all boundaries between disciplines are man-made, but highly influence the way humans interact with their environment. A key point each modeler should be aware of is that any property of a model that is a result of one of his own choices, should not affect the results of the model. Examples of modeler's choices are: relevance of time and space scales, references, system boundaries, domain boundaries, coordinates and metrics. If a variation in one of these choices leads to completely different conclusions about the problem for which the model is constructed, the model obviously does not serve its purpose as it tells us more about the modeler than about the problem to be solved. Again, when the issue is phrased in this manner, it is hard to disagree, but practice shows that this modeling principle is often violated.

Modeling decisions should be strongly backed with a good understanding of the problem situation and sufficient domain knowledge. The way how an original system is decomposed by modelers determines model composition and interaction. The level of resolution, or when to stop recursively decomposing a system, again depends on the purpose of analysis. We aim at decomposing a system into less complex parts and interacting relations such that they capture the essence of a system to serve the intended simulation goal and at the same time provide certain flexibility for other composite combinations in order to represent a class of systems in a certain domain. This naturally requires modelers to consider the class of possible applications of the prospective model components instead of only one application.

A reasonable question to ask is what are the similarities and differences among the potential target systems or what are their possible changes. This kind of decomposition criteria is advocated as early as in PARNAS (1972) in a more systems design oriented context. The author argues that one should begin with a list of different design decisions

[15] TOLK et al. (2010) have discussion on the interoperability and standardization of model components.

or design decisions which are likely to change, and then design each component to hide such a decision from the others (ibid.). The word "hide" should not be understood in a narrow sense of encapsulation in object-orientation. As we will try to show later in this chapter, there are many ways that could provide flexibility by the design of model components to "hide changes".

Source of Rail Domain Knowledge In developing LIBROS, which we use for cases in this chapter, part of the domain knowledge is obtained from well established theories (in rail and public transport) presented in books[16] and articles such as those from the TRB (www.trb.org) journal and meetings. During the research, formal and informal meetings were conducted regularly with experts from or affiliated to HTM. The author was present at the organization on a weekly basis, visited different departments (e.g., planning, operation, control, maintenance, infrastructure, depot and ICT), met and interviewed specialists from these departments, and collected internal documents when available.

According to PACHL (2002), a railway system consists of three essential elements[17]:

1. the *infrastructure* with the trackwork, signaling equipment, stations, etc.;
2. the *rolling stock* with locomotives and cars; and
3. the system of *operating rules and procedures* for a safe and efficient operation.

This provides a guideline in how to decompose a railway system. For example, we need to model things such as rolling stocks (hereafter called vehicles), tracks, signals and different operating procedures. There are specific modeling choices that need to be decided upon.

4.2.1 Modeling Vehicles

The first modeling choice concerns *whether we represent the vehicles as inputs and outputs of some other models or represent them as models that have autonomous behavior*. More specifically, in the former case, the vehicles are to be represented as messages (or events) being transmitted from one model, e.g., infrastructure, to another (LEE et al. 2004, WAINER 2006). In the latter case, they are to be represented as models that can act on their own. We choose to do the latter basically for the following reasons.

A rich vehicle representation Represented as a model component, a vehicle can have dynamic autonomous behavior. This eases modeling driving-on-sight and allows for a rich representation of vehicles as required by the modeling needs discussed in § 4.1.1.

[16] The major domain text books used are the following:
- Railway Operation and Control (PACHL 2002)
- Railway Timetable & Traffic: Analysis-Modelling-Simulation (HANSEN and PACHL 2008)
- Railway Signalling & Interlocking: International Compendium (THEEG and VLASENKO 2009)
- Urban Transit: Operations, Planning, and Economics (VUCHIC 2005)

[17] The first two are regarded as the "hardware" and the third as the "software" of railways (PACHL 2002).

Modularity and extensibility Message representations are in principle static; i.e., a message does not have behavior per se. Although behavior may be mimicked by modifying the message content, the modification is dependent on the behavioral units (i.e., atomic models in case of DEVS) that hold the message. This design choice raises dependency as opposed to modularity. Furthermore, when we wish to change or extend the vehicle behavior, we would have to change the aforementioned behavioral units instead of the vehicle "itself". This affirms the advantage of the choice from the aspect of modularity and extensibility through *separation of concerns*. Their importance are discussed in § 4.1.1 and § 4.1.2.

A follow-up question is *whether we model the locomotives and cars separately*. We choose not to do so because such a separation is not required by the intended model use. The vehicle model in LIBROS thus has "an inseparable body". This solution is however open for extension. When a separation is needed[18], the vehicle model can be used as "a locomotive" without change, and a car model may be added.

Choosing a simple but modular design is a principle we used often in LIBROS component design. Some other examples are: how to choose units and boundaries of infrastructures, weather and how to separate operating rules and infrastructures, some of which we will soon discuss.

Note 1 *Among competing modeling choices in model component design, choose a simple solution that is modular so that it can be extended when needed.*

Modeling vehicle movement is the base element to estimate running time (HANSEN and PACHL 2008). It calculates the speed-distance profile of a vehicle traveling from one point to another (LI and GAO 2007). Different modeling styles (continuous, discrete time or discrete event abstraction) can be used; each has pros and cons in terms of model detail, accuracy, modularity, and computational efficiency[19].

A basic idea in designing the vehicle model in LIBROS is that the movement is represented by a changing relation of the vehicle model to where the vehicle is situated in

[18] A separation may be needed when modeling, e.g., operations of *marshaling yards*, which is for the marshaling of trains, exchange of wagons between trains and splitting up of trains (THEEG and VLASENKO 2009).

[19] A continuous abstraction of vehicle behavior is obtained by assuming a continuous time base and defining the rate of state (e.g., location, speed and acceleration) changes of a vehicle using differential equations. A numerical integrator executes the model. Some examples of continuous vehicle models are discussed in HANSEN and PACHL (2008).

A discrete time abstraction of vehicle movement is obtained through the definition of a time-invariant recurrence relation between the current state of a vehicle and the successive state after a predefined time interval has elapsed. Discrete time models of vehicle behavior are developed, e.g., with difference equations and cellular automata (LI et al. 2005, LI and GAO 2007).

A discrete event abstraction of vehicle movement is obtained through the definition of events which trigger significant state changes of a vehicle being modeled. The choice and definition of events are based on the purpose of model use and the interest of modelers. Some examples of discrete event vehicle models are presented by MIDDELKOOP and BOUWMAN (2001), LU et al. (2004) and LI et al. (2009).

The choice of (the length of) the time interval or integration step in the discrete time or continuous modeling approach is basically a trade-off between computational efficiency and simulation accuracy. The smaller the time step, the more accurate and more computationally demanding. Owing to a longer tradition, the continuous modeling style of vehicle movement appears to be the most intuitive. Well known equations such as the equations of motion can be directly applied without much further modeling effort.

a rail network. We need to design how a vehicle model communicates with its environment (including the *infrastructure* and other *vehicles*) and decides on its movement accordingly. To proceed, we next discuss some concepts related to modeling infrastructures.

4.2.2 Modeling Infrastructures

Many factors in the railway infrastructure influence vehicle movement, e.g., track curvatures, control signals and speed restrictions. It is best practice in the modeling of railway infrastructure to use structures derived from *graph theory* (HANSEN and PACHL 2008). We can decompose the rail network into parts and represent the rail network as a directed graph.

A choice here is *whether we represent the rail network as a data structure or as simulation models*. A data structure representation of a rail network would be an infrastructure map or graph *without* behavior, while a simulation model has behavior. The alternatives of the choice need further elaboration. In the former case, a behavioral unit needs to hold the infrastructure map and perform the appropriate actions. We may, e.g., have a design of letting each vehicle know the complete network.

1. Each vehicle holds an infrastructure map and knows its own locations. It announces itself to other vehicles so that it can be "seen".

This option, Figure 4.2–1, results in strongly connected (vehicle) components which need broadcast like communications. This solution is *not scalable* with regard to the growth of map size and vehicle number. A direct improvement of this is to use a centralized solution.

2. A behavioral unit — let us call it a coordinator — holds the infrastructure map, and maintains a list regarding the vehicles and their locations. The coordinator communicates with the vehicles and informs them about the situation of their environment.

In this option, Figure 4.2–2, all vehicles are connected to one coordinator instead of to each other. This reduces the communication cost, i.e., $1 : n$ as opposed to $n : n$ in the previous option. But the vehicle models are now largely dependent on the coordinator and the latter is a singleton component that is loaded with *heavy duty*. It needs to, e.g., maintain the states and positions of the vehicles, find which vehicles may affect the others, and inform the potentially affected vehicles so that they may adapt their movement accordingly.

Although the discrete event approach is often the most efficient among the three modeling style (ZEIGLER and LEE 1998, GIAMBIASI et al. 2000, KOFMAN 2003a), it requires a careful analysis and design of the simulation model. A good design of a discrete event representation of a continuous or hybrid system can render simulation more efficient than using discrete time or continuous representations. However, using the discrete event modeling style per se does not guarantee efficient simulation.

In both ways, we have not yet mentioned that the infrastructure also has behavioral parts. The sensors, switches, and signals in a rail network are non static[20]. They "work" together in some areas in the rail network to safeguard the vehicle movement. They too need to interact with the vehicles. Similar to what is described in option 2, this interaction can be managed through the coordinator.

2*. A coordinator also holds a list of the behavioral infrastructure parts and their locations in the map. It manages the communications among these infrastructure parts and the vehicles.

This, Figure 4.2–2*, results in a more complex singleton component which has to handle all the communications among the vehicles and dynamic infrastructure parts (and to detect communication relations among them). To reduce the complexity of the coordinator, we can again use the principle of *separation of concerns*: divide and distribute the responsibilities, for instance, to some sub-coordinators. An intuitive choice of the division would be a "geographical partition" in which a sub-coordinator is responsible for one partitioned geographical area. In this regard, the communications may still be handled through a coordinator or only by the sub-coordinators themselves. As such, we may have the following two options when we partition a rail infrastructure network into some sub-areas.

3. Each sub-coordinator holds a sub-map and manages a partitioned area. The sub-coordinators communicate with a higher level coordinator who has a global view of the vehicle locations.
4. Each sub-coordinator holds a sub-map and manages a partitioned area. It has a local view of the vehicle locations. The sub-coordinators communicate with one another.

At first glance, the difference between the two options may be only the *degrees of centralization* in communications. For example, in option 3, Figure 4.2–3, vehicles communicate with each other through the coordinator (as in option 2 or 2*), and they communicate with the dynamic infrastructure parts through sub-coordinators[21]. In option 4, Figure 4.2–4, the central coordinator is left out. Each sub-coordinator shall locally handle the communications among the vehicles and the dynamic infrastructure parts within its area. When two vehicles in two partitioned areas want to communicate, the two corresponding sub-coordinators shall together handle the communications.

An important difference between the two options, however, is that option 3 has a static model structure while option 4 would require *dynamic model structure*. A model having a dynamic structure is a model that is designed to change its structure during

[20] Sensors are used to refer to different vehicle detection and track clear detection devices used in rail operations and controls (PACHL 2002, THEEG and VLASENKO 2009). Switches (also called switchpoints or points) are movable track elements that are used to transfer rolling stocks from one track to another (THEEG and VLASENKO 2009). Signals indicate if a movement may enter the section of track behind (i.e., beyond) the signaling equipment (PACHL 2002).

[21] Without a central coordinator, we may directly connect each vehicle model to each sub-coordinators. But this setting of connections may be less desirable.

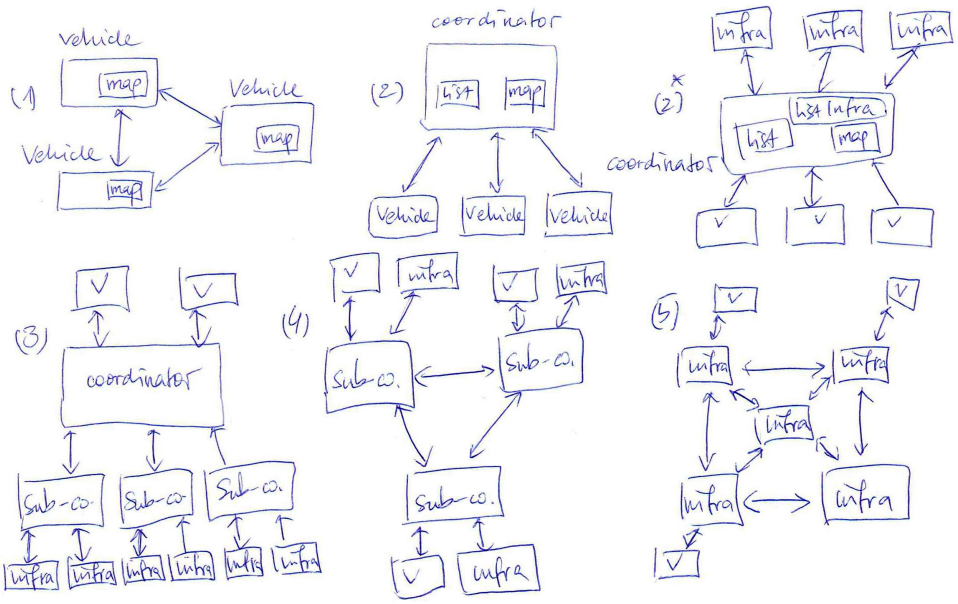


Figure 4.2: Modeling options for infrastructure models

simulation. The dynamic structure in option 4 is related to the vehicle models. As they "move" in the infrastructure network, they need to be connected to the corresponding sub-coordinators when appropriate. Since only the sub-coordinators at the adjacent areas need to be connected to each other, a meaningful partition of the infrastructure network in option 4 can lead to a simpler structure of sub-coordinators that is homomorphic to the infrastructure network.

The vehicle models in option 4 are connected to the sub-coordinators instead of one central coordinator, which is a less centralized solution compared to option 3. We may take this *decentralization* one step further by leaving out the sub-coordinators. The idea is that vehicle models can be directly connected to infrastructure models. In the previous modeling options, the infrastructure network is represented by maps or sub-maps (i.e., data structures) in which only the behavioral parts are represented by infrastructure models. In other words, the infrastructure is represented partly static and partly dynamic (which is indeed the case in reality). We can, however, represent the infrastructure solely with (dynamic) models to achieve *uniformity* in infrastructure representation. In this way, the connectedness of the infrastructure parts itself represents the network structure. As in option 4, the vehicle models can be dynamically connected to the infrastructure models in order to model their movement.

5. The infrastructure is represented by a network of connected infrastructure mod-

els. The vehicle models are dynamically connected to the successive infrastructure models in the moving trajectories to represent the vehicle movement.

We are in favor of this modeling concept, Figure 4.2–5, firstly for the uniformity in infrastructure representation. This uniformity increases *cohesion* in the affected models and potentially improves their composability. Second, the coordinators or subcoordinators are left out in this design, which reduces the *number of artifacts* that are constructed purely for the purpose of modeling. This potentially leads to simpler and more understandable morphisms. Third, the *decentralized communications* allow for convenient modeling of *autonomous* behavior. Given the requirement in vehicle modeling, this infrastructure representation is a suitable choice. Fourth, representing the infrastructure network as connected models permits *flexibility* in the sense that modelers may change model behavior by changing (infrastructure) model structure. As we will show in § 5, this flexibility is positively accounted for in model generation.

Note 2 *Using less artifacts which are constructed purely for the purpose of modeling potentially leads to simpler and more understandable morphisms.*

Note 3 *In modeling communications, when a decentralized design results in strongly or almost strongly connected model components and broadcast like communications, centralization can be a beneficial alternative in terms of communication cost. The latter, however, can be less convenient in modeling the autonomy of aforementioned components.*

As discussed in § 4.1.1, a microscopic model is needed for running time calculation, timetable assessment and rail operational simulation. The granularity of the infrastructure model in LIBROS is in accordance with a common sense decomposition of the engineered system. At its lowest description level, the rail infrastructure model is a network of *Rail Infrastructure Elements* (RIEs) such as track segments, sensors, switches and signals (the last three are hereinafter abbreviated as 3S). The RIEs can be used to recursively compose higher-level infrastructure components[22]. This recursive composition makes up an infrastructure network that has a (multilevel) *Compositional Containment Hierarchy* (CCH)[23]. For example, a tram stop (or halting place) model is composed of track segments and a sensor; an intersection (or crossing) is composed of track segments and 3S; and the two composed models can further be composed.

In controlled areas such as an intersection (or crossing), the 3S models need to communicate with one another. More specifically, the signals (or signaling), e.g., at an intersection, need to be coordinated for safety control, during which the sensors in the area are for vehicle detection, and the switches change positions if necessary in order to allow vehicles to move from one track to another. This coordination (or control) is carried out by a control unit in the area. The use of (locally) centralized communications (and control) through the control unit is *a logical outcome of informed modeling rather than a deliberate design choice* between centralized or decentralized communications as previously discussed.

[22] In DEVS, the former would be specified in atomic models and the latter in coupled models.
[23] We have more explanation on this in § 5.1.1.

4.2.3 Modeling Vehicle Communications

A vehicle model needs to communicate with its "environment" in order to decide upon its own actions. The discussion in the previous section presents a general concept of representing the infrastructure as a network of connected infrastructure model parts, and connecting vehicle models dynamically to the infrastructure models where they are located. This section discusses this design in more detail. We address two issues:

i. How to connect a vehicle model with an infrastructure model?
ii. How does a vehicle model communicate with another model?

Suppose the following simple scenario: *a vehicle drives along a rail track composed of four track segments.* A model of this would have four successively connected track segments ($T1 \sim T4$), as shown in Figure 4.3–1; a vehicle model (V) can be connected to $T1 \sim T4$ one after another at four time instances corresponding to the time[24] that is needed by the vehicle moving from one track segment to the next.

Suppose that $T2$ *and* $T3$ *together form a composed infrastructure model* T'. The model V can be connected with $T2$ or $T3$ in two ways:

(a) place V in the composed model T', and connect V directly with $T2$ or $T3$, as shown in Figure 4.3–2; or
(b) place V outside of T', connect V with T' at port p, and connect port p with $T2$ or $T3$, as shown in Figure 4.3–3.

Which way is better? We cannot yet decide as such. Let us take a look at both ways how a vehicle model can communicate with another model. Suppose now *there is another vehicle model* (V') *connected to* $T1$. How can V communicate with V'? Corresponding to the previous two ways of connection, we can also connect the two vehicles in two ways:

(a) connect V with V' through a port of T', as shown in Figure 4.3–4 as an outcome of 4.3–2; or
(b) directly connect V with V', as shown in Figure 4.3–5 as an outcome of 4.3–3.

Suppose again that *at the position of* $T4$ *there is a signal* (S) *instead and we want to connect* V *with* S. The resulting connections would be similar to those that connect V and V' in both ways.

Apparently, neither way is insofar better than the other in terms of easiness of connection. Both would have direct and indirect connections[25] of model components depending on whether the components have the same parent model. As the vehicles in principle can be anywhere in the infrastructure network which has a CCH, we often need to connect (and disconnect) models that belong to different levels. These connections can be costly.

[24] This time is calculated, e.g., based on the length of the track segment and the vehicle's speed and acceleration.
[25] Two components are indirectly connected when there are more than one link connecting them. A link (or connection) directly joints two ports of two components.

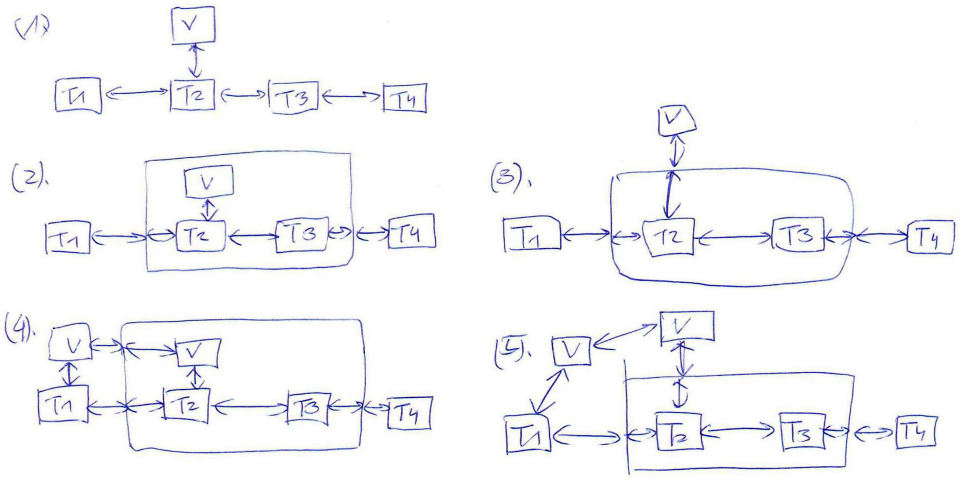


Figure 4.3: Modeling options for vehicle and infrastructure model connections

In order to further our discussion on the design of vehicle communications, we shall first look at with whom the vehicle models communicate. Basically vehicle models need to communicate with RIE models (i.e., vehicle-to-infrastructure or V2I) and with other vehicle models (i.e., vehicle-to-vehicle or V2V). We choose to *focus on the most relevant* meaning that a vehicle model at a certain instant only communicates (1) with its next closest RIE, *and* (2) with its closest preceding and following vehicles (if any) within a distance range. For example, when a vehicle is approaching an intersection, the movement is computed based on the distance and the state of the signal at the intersection; if there is another vehicle ahead, the vehicle needs to adapt its speed to avoid collision. Since a vehicle's location is not static, neither are its connections to its next closest RIE and its preceding vehicle (i.e., the V2I and V2V connections). Our interest in reducing the cost of dynamic connections naturally brings out the following question: *can we use static connections for vehicle communications?*

An aforehand setup of all possible connections is not an acceptable option. There is, nonetheless, an option worth considering. The infrastructure model is a (statically) connected network. When a vehicle model is connected to the network, it practically has connections to any models that are as well connected to the network, despite that the connections are mediated in the sense that there are RIE models in-between. The basic idea is that: *we can use the infrastructure network as the backbone of vehicle communications.*

Take Figure 4.3–4 (p. 74) as an example. In this setting, vehicle V' is connected with V through track segments $T1$ and $T2$. It is possible to establish communications between V' and V through the two intermediates without setting up new *dedicated* con-

nections between the two vehicles. When V' needs to send a message to V or vice versa, $T1$ and $T2$ shall pass on the message.

Following this principle, any vehicle model can communicate with any other vehicle and RIE via the infrastructure network. We only need to dynamically connect each vehicle to the RIE model corresponding to where the vehicle is located. In this case, it makes sense to directly connect a vehicle with a RIE, as shown, e.g., in Figure 4.3–2 and 4. The number of direct dynamic connections, therefore, is equal to the number of vehicles in the simulation model. We call this mediated message passing mechanism *message propagation. How is message propagation compared to communications using dedicated dynamic connections?* The next section tries to answer this question.

4.2.3.1 Dedicated Dynamic Connections vs. Message Propagation

In the case of *Dedicated Dynamic Connections* (DDCs), point-to-point V2V and V2I connections are established through which vehicles can exchange messages *directly* with the other vehicles or RIEs. As stated, when one vehicle needs to compute its movement at a certain instant, it needs to know its next closest RIE and preceding vehicle (if any)[26]. Each of the two communications at a certain instant generally requires three successive steps:

(1) given a vehicle model, search for the next closest RIE or preceding vehicle;
(2) if found, establish a connection (or two connections if the connection is directed) between the two models;
(3) and finally the two models can communicate.

In the case of *Message Propagation* (MP), the existing static connections of the infrastructure network are used, and the message are exchanged *indirect* through the mediated RIE or RIEs. As such, new connections are not needed (step 2 above) between the pairs of communicating models. This gives a *simpler model structure*. We can also spare the search function (step 1 above) since the next closest RIE and preceding vehicle can both be found during MP. In other words, the process of MP combines steps 1 and 3 above. The basic concept of MP can be simplified as following:

(1) at a certain instant, when a given vehicle model needs to know its next closest RIE and preceding vehicle, it sends out a (request) message;
(2) the message is forwarded by the RIEs along the vehicle's route;
(3) once the next closest RIE or preceding vehicle (if any) receives the message, it sends a (response) message back;
(4) the message is forwarded by the RIEs back to the original vehicle.

After such a round of MP, a given vehicle model would receive at least one and at most two response messages. To achieve the same result, the DDCs (of V2V and V2I) need to run the steps 1∼3 for twice. However, there is a tradeoff. The MP introduces

[26] A vehicle always has a next closest RIE, but not necessarily a preceding vehicle.

	Search	*Connections*		*Communications*	
	BFS	*LCA*	*Setup*	*Message sending*	*Transitions*
Dedicated Connections	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log(n+v\log n))$	$O(\delta_c)$
Message Propagation	-	-	-	$O(\log n)$	$O(\delta_p \log n)$

Table 4.1: Time complexity per V2I or V2V communication with *dedicated dynamic connections* or *message propagation*. The complexity denoted is per unit, which estimates how much total communications cost is required for *one vehicle movement computation* at one instant. The complexity in a whole simulation run raises over the total number of vehicles generated v_T and the number of RIEs n, e.g., the total time of BFS in a simulation run is $O(v_T n \log n)$.

the overhead of message forwarding by the mediated RIEs; each message forwarding entails one extra model state transition.

In order to make an informed decision, the time complexity and space complexity (see, e.g., KNUTH 1997, CORMEN et al. 2001) of both designs of communications are estimated and compared[27]; see Table 4.1. The estimation uses one vehicle movement computation as *one unit*. If not otherwise stated, the complexity hereinafter presented is per unit, which estimates how much communications cost is required for *one* movement computation at *one* instant, given that for each computation a vehicle needs to know its next closest RIE and preceding vehicle.

Cost of Dedicated Dynamic Connections In the case of DDCs, step 1 can be solved by a *Breadth-First Search* (BFS) in the infrastructure network[28] starting from the RIE node where the vehicle model is located. It is well known that BFS has linear time and space complexities, denoted as $T(f_{\text{BFS}}) = O(n)$ or $f_{\text{BFS}} \in O(n)$ (see, e.g., CORMEN et al. 2001)[29]. Since the vehicles drive along predefined routes (i.e., paths), we can improve the search cost to $O(\log n)$. In both cases[30], n is the number of nodes (or vertices), i.e., RIEs, in the graph.

For step 2, setting up (or removing) a dedicated point-to-point connection between two models requires the knowledge of the common parent of the two models. This can be solved by a *Lowest Common Ancestor* (LCA) finding algorithm in the CCH[29] of

[27] We are aware that high level designs can be concretized in many different ways even with the same modeling concept in mind. Our estimation is theory based and is deemed appropriate since no detailed model design is available at a high level design stage.

[28] The infrastructure network model, as how it is connected, is a *directed graph* with RIEs as nodes. The infrastructure network model, as how it is composed, however, has a Compositional Containment Hierarchy (CCH). We have more explanation on CCH in § 5.1.1.

[29] In literature, the BFS runs in $O(n+m)$ time where n is the number of nodes and m is the number of edges in a graph. We consider it to be $O(n)$ as the infrastructure network is sparse, i.e., $n \sim m$.

[30] In order to keep the discussion as general as possible, search improvement with bound constrains is not considered, as such bound modifications are not always applicable and are different from case to case.

the infrastructure model. The LCA has as well linear time and space complexities, but to the tree height (see, e.g., ALSTRUP et al. 2004). The number of RIE nodes n in the infrastructure network is the leaf number in the tree of CCH (hereinafter called *model composite tree* or MCT). The number of inner nodes in relation to the leaf number n in a full k-ary tree is $(n-1)/(k-1)$; hence the number of total nodes in a tree is $2n-1$ in the worst case[31], which has a tree height of $\log(2n-1)$. The LCA, therefore, has a complexity of $\mathrm{O}(\log n)$, i.e., $T(f_{\mathrm{LCA}}) = \mathrm{O}(\log n)$.

The connection setup (or removal) basically has the same time and space complexities as LCA since it uses the (intermediate and final) result of the latter. To set up the connection we first need to create ports. The ports and connection themselves create extra space complexity of $\mathrm{O}(\log n)$ for each connection. Because each vehicle always has one or two connections during its (simulation) lifetime, the total space complexity of the connections at a certain instant is $\mathrm{O}(v \log n)$ where v is the vehicle number in a simulation at an instant. Note that v is smaller than v_T which is the total number of vehicles (generated) in a simulation.

In step 3, the two models can communicate with each other. When a message is sent, the reception of the message triggers the state transition of the receiver. Message sending requires a search in the connections list where the existing vehicle connections are stored. As just mentioned, the space complexity of the connections at an instant is $\mathrm{O}(v \log n)$. We hence estimate the time complexity of message sending to be $\mathrm{O}(\log(n + v \log n))$. The space complexity of one message is constant, i.e., $\mathrm{O}(1)$. In using DDCs, since there is only one receiver involved, the cost of the state transition is dependent on the receiver model. We denote the transition function as δ_c (where c stands for connection). See below for the discussion on $\mathrm{O}(\delta_c)$.

Note 4 *Configuration (and reconfiguration) cost is a criterion in choosing communication mechanisms of model components that have dynamic structures.*

Cost of Message Propagation MP relies on indirect communications. This has advantages as well as disadvantages. An obvious advantage is that extra dynamic connections are not required for the intended communications. Hence, the related search and connection costs for (re-)configuration are spared. Message *sending* itself is easy because one vehicle is only connected to one RIE at any instant. The MP time complexity grows with the distance, i.e., the number of intermediary RIE models along the route, between the original sender and the final receiver, i.e., $\mathrm{O}(\log n)$.

For message *passing*, we have to impose *overhead* on the intermediary RIE models. Upon the reception of a message, each intermediary RIE model needs to forward the message. Once the message reaches its final receiver, be it a RIE or vehicle model, the receiver responds to the message and, if necessary, computes its state accordingly. In both cases, state transitions are required. We denote the transition function as δ_p (where p stands for propagation). An estimate of the total cost of the transitions is $\mathrm{O}(\delta_p \log n)$ as each model along the route is engaged.

[31] A tree has a maximum node number of $2n-1$ when it is binary, i.e., $k=2$, given a leaf number n.

On Transitions δ_c and δ_p From the perspective of a sender or receiver, the outcome of DDCs and MP (through functions δ_c and δ_p) should be comparable. In the case of DDCs, a sender and a receiver communicate directly. In a modular design, they are supposed to exchange information about their local states. The situation between the two is not contained in the message per se. When latter information is needed, extra communication is required. This extra cost shall be accounted for by δ_c.

In the case of MP, the models along the propagation route are also involved. Each can participate as a communicating party rather than merely a messenger. Hence, δ_p can be assigned with tasks beyond simple message forwarding. Modelers may design rich behaviors that the intermediary models shall perform so long as these behaviors are conceptually consistent with the purpose of the model components. In the context of LIBROS, the intermediary RIE models are designed to perform the aforementioned search function and to enrich the information contained in a message. To conclude, the state transitions δ_p performed by the models in MP shall:

(a) fulfill the functionality of transitions δ_c in DDCs, and

(b) undertake the overhead Λ introduced by the use of MP.

MP can be a good choice when the function δ_c in DDCs can be replaced by simple tasks performed by δ'_p of the models in MP even after taking on the overhead Λ; that is when $\mathrm{O}(\delta_c) \gtrsim \mathrm{O}(\delta_p \log n) \backsimeq \mathrm{O}((\delta'_p + \Lambda)\log n)$. In principle, $\mathrm{O}(\delta_p)$ can be kept low by low $\mathrm{O}(\delta'_p)$ and $\mathrm{O}(\Lambda)$, meaning that a good decomposition of function δ_c into δ'_p and a good design of the transition functions which handle the overhead Λ. In LIBROS, the transition function δ_p in the RIE models have a constant time complexity, i.e., $\delta_p \in \mathrm{O}(1)$ for each message forwarding. This makes MP an attractive design option.

4.2.3.2 A Joint Consideration

After a number of considerations, we choose to use MP for the modeling of vehicle communications in LIBROS. We are in favor of MP, however, not only because it has a reasonable cost estimate in terms of computation (which is just discussed and shown in Table 4.1) given our modeling context. The choice of MP in LIBROS is consistent with the three previous notes stated in § 4.2.1 (p. 68) and § 4.2.2 (p. 72), and it has advantages in terms of model design itself.

Simpler model structure The indirect communications in MP ask for a simpler model structure compared to that in DDCs since dedicated connections are not required. A direct outcome of this is that it needs less couplings, whose benefits are discussed in § 4.1.2.

Simpler model components The need of less functionality (in case of BFS, LCA and connection setup) or the decomposition of large functionality into simpler ones (in case of δ_c and δ_p) means that less model components or smaller components are to be designed. Simpler components are easier to understand, develop, test and document. They are also easier to be reused and extended, which are

desirable features discussed in § 4.1.1.

4.3 Model Design in LIBROS

4.3.1 A Communication Mechanism: Message Propagation

MP in principle is a mechanism for *decentralized indirect communications in a connected network model*. Designing communications should respect the corresponding systems knowledge (§ 2.1.3 and § 2.1.4) while considering how components could communicate with each other efficiently with respect to modularity, strong cohesion and weak coupling (§ 2.3 and § 4.1.2). Connecting locally each vehicle model with the RIE model where the vehicle is located as discussed in § 4.2.3, is an effort to decrease dynamic couplings and to increase cohesion in the couplings.

As discussed in § 4.2.2, the rail infrastructure model in LIBROS, at the lowest description level, is a network of RIEs such as track segments and 3S. For MP, each RIE is designed such that it is capable of *message propagation*, which can be in the direction of the traffic or in the opposite direction.

Basic Concept A vehicle model's main task in LIBROS is to correctly compute its movement based on the information about its next closest RIE and the preceding vehicle it is approaching. In using MP, if either type of information is absent before a movement, a vehicle sends a message forward[32] which requests this information. The vehicle model's next closest RIE and preceding vehicle responds to the message, which is then propagated back to the original vehicle. Once obtaining the information, the vehicle is able to compute its movement trajectory to reach the next infrastructure. After having reached the RIE, a new iteration of this process starts again until the vehicle reaches its final destination.

MP is also used when vehicles or RIEs change (or update) their states and want to make the state changes known to some other (vehicle) models. A vehicle can change its state during a movement trajectory when its velocity changes. A RIE model such as a signal changes its state when its signaling sign changes. For such cases, a model announces its state change by sending a message backward[33]. In this way, an approaching vehicle model would receive the message and change its movement accordingly when necessary.

Example 4.1 *Figure 4.4 illustrates a simple model example composed of four successively connected track segments (T0 ∼ T3) and two vehicles (V0 and V1). Each track segment has a length (L0 ∼ L3) and a speed limit (SL0 ∼ SL3). Each vehicle has a vehicle length (VL0 and VL1), its position (P0 and P1) on the track segment[34] it is coupled with, and its*

[32] When a message is sent forward, it is propagated in the direction of the traffic.
[33] When a message is sent backward, it is propagated in a direction against the traffic flow.
[34] A vehicle's position on a track segment is defined as the distance between the vehicle's front end and the start node of the track segment.

current speed limit (CSL0 and CSL1). Suppose that V0 is on T0 and V1 is on T3, i.e., V0 is behind V1 given that the direction of the traffic is from left to right; and V0 does not have the information about its next infrastructure nor about its preceding vehicle.

4.3.1.1 Request and Response Messages

Given the situation in the example, $V0$ would send a message forward $(\overrightarrow{M0})$[35] to request information. Two message prorogation sequences are generated upon this action:

(1) $\overrightarrow{M0}, \overrightarrow{M1}, \overrightarrow{M2}, \quad \overleftarrow{M3}, \overleftarrow{M4}, \overleftarrow{M5},$ and

(2) $(\overrightarrow{M0}, \overrightarrow{M1}, \overrightarrow{M2},) \overrightarrow{M3'}, \overrightarrow{M4'}, \quad \overleftarrow{M5'}, \overleftarrow{M6}, \overleftarrow{M7}, \overleftarrow{M8}, \overleftarrow{M9}.$

In our original design for MP, a closest RIE would respond to a request message (originated from a vehicle). As such, $T1$ would respond to $V0$'s request in the example; and $V0$ should subsequently calculate its movement until $T1$. This means that for each vehicle, each RIE would require one round of MP. In order to reduce the frequency of MP and of the consequent model transitions, the following rule is defined: *a RIE model responds to a request message only when the RIE requires or potentially requires a change in movement of the requesting vehicle.* We call such a RIE the *next closest RIE of interest* (NCRI) of a vehicle. Here are some examples such RIEs:

(a) the RIE has a defined speed limit which is different to the current speed limit of the requesting vehicle;

(b) the RIE is a signal;

(c) the RIE is (or is in) a stop, a station, a terminal, etc.

Referring back to Example 4.1, in MP sequence (1), we assume that

$$(SL1 = \infty \vee CSL0 = SL1) \wedge (SL2 \neq \infty \wedge CSL0 \neq SL2)$$

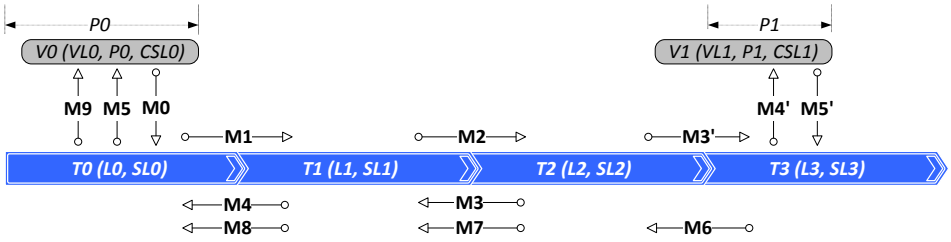


Figure 4.4: Message propagation in Example 4.1

[35] We use an arrow over a message to indicate its direction forward (→) in case of a request message or backward (←) in case of a response message.

It means that the speed limit of $T1$ is undefined, i.e., there is no speed limit ($SL1 = \infty$) or it is the same as the current speed limit of $V0$ ($CSL0 = SL1$), and the speed limit of $T2$ is defined ($SL2 \neq \infty$) and it is *not* the same as the current speed limit of $V0$ ($CSL0 \neq SL2$). This condition coheres with the example (a) given above. Therefore, $T0$ and $T1$ do not respond to the request message but propagate it forward ($\overrightarrow{M1}$ and $\overrightarrow{M2}$); $T2$ responds to the request originated from $V0$ by a backward message $\overleftarrow{M3}$ which is propagated back to $V0$ through $T1$ and $T0$ ($\overleftarrow{M4}$ and $\overleftarrow{M5}$).

While responding to the request from $V0$, $T2$ also propagates the request further forward to $T3$ ($\overrightarrow{M3'}$) because $V0$'s *preceding vehicle* (PV) is not yet found. This generates MP sequence (2). For MP, there are two other rules defined: *a RIE model forwards a request message to the vehicle model closest to its start node when there is any vehicle coupled with it; when a vehicle model receives a request message, it responds unconditionally.* In the example, $T3$ has $V1$ coupled with it, so it forwards the request to $V1$ ($\overrightarrow{M4'}$). In response, $V1$ sends back a message ($\overleftarrow{M5'}$), addressed to $V0$, which is propagated back to $V0$ ($\overleftarrow{M6} \sim \overleftarrow{M9}$). By the time, $T3$ stops propagating the message further because both the NCRI and the PV of $V0$ are found. This concludes this round of message propagation, which is initiated by $V0$ sending out a request message $\overrightarrow{M0}$ and ended by two response messages $\overleftarrow{M5}$ and $\overleftarrow{M9}$:

(1) $\overleftarrow{M5}$ is originated from $T3$, $V0$'s NCRI, and

(2) $\overleftarrow{M9}$ is originated from $V1$, $V0$'s PV.

In MP, the messages (thirteen in total in Example 4.1) shall contain sufficient information so that they can be propagated or used correctly. The following information is included in the messages in LIBROS:

(i) the message type, e.g., forward, backward;

(ii) the original sender's ID and type;

(iii) the information about the original sender, i.e., the values of the sender's state variables at the instant of message sending, e.g., when the sender is a vehicle, this can be its speed, acceleration, etc.; when the sender is a signal, this can be its signaling state;

(iv) the final receiver's ID (if applicable);

(v) the distance between the original sender and the receiver.

4.3.1.2 Distance Accumulation in Messages

The (v) distance between the original sender and the receiver is accumulated by the vehicle and RIE models during MP. Table 4.2 gives an example of distance accumulation step by step. As already illustrated in MP sequence (2) of Example 4.1 (Figure 4.4, p. 80), in response to $V0$'s request, message $\overleftarrow{M5'}$ is created by $V1$, addressed to $V0$, and sent to $T3$. The distance d is counted backwards starting from the *rear end* of a vehicle

sender. *When a vehicle model creates a backward message, the distance contained in the message is set to the value of the vehicle's position on the RIE model with which it is coupled, deducted by the vehicle's length.* Thus, d in $\overleftarrow{M5'}$ is of value $P1-VL1$.

When a RIE model receives a backward message, it does not change the value of the distance if the original sender is a vehicle that is coupled to the RIE; otherwise the RIE increases the value by its own length. Following this rule, $T3$ does not need to change d in $\overleftarrow{M6}$, while $T2$, $T1$ and $T0$ add their own lengths respectively to d in the three successive messages ($\overleftarrow{M7} \sim \overleftarrow{M9}$). Since $\overleftarrow{M9}$ is addressed to $V0$ which is coupled with $T0$, $T0$ forwards it to $V0$. This action concludes this backward propagation.

As a result, $V0$ holds the response $\overleftarrow{M9}$ with $d = P1-VL1+L2+L1+L0$ which is the distance from the rear end of $V1$ till and include $T0$. When $V0$ needs to know its distance to $V1$, it simply shall deduct d with its own position on $T0$.

4

4.3.1.3 Update Messages

As mentioned, MP is not only used for requesting information and the subsequent response, but also for announcing state updates by a vehicle or RIE model. Suppose that, in Example 4.1 (Figure 4.4, p. 80), at a certain instant, $V1$ did not receive any request message but just had a state update for some reason. In order to announce its state update, $V1$ creates an update message, which will be propagated backward to the most relevant, $V0$, the vehicle driving behind $V1$. As expected, the MP sequence and the distance accumulation for the update message would be the same as those described for the response message in the example.

An update message is created, without an addressed recipient, by a vehicle or RIE model straight after an state transition, and is propagated backward to a vehicle model closest to the original message sender within a predefined distance bound. The details of message propagation rules can be found in § B.1.2.

	Sender	Distance Accumulation	Message	Distance value (d) in Message	Receiver
1.	$V1$	$d \leftarrow P1-VL1$	$\overleftarrow{M5'}$	$P1-VL1$	$T3$
2.	$T3$	d	$\overleftarrow{M6}$	$P1-VL1$	$T2$
3.	$T2$	$d \leftarrow d+L2$	$\overleftarrow{M7}$	$P1-VL1+L2$	$T1$
4.	$T1$	$d \leftarrow d+L1$	$\overleftarrow{M8}$	$P1-VL1+L2+L1$	$T0$
5.	$T0$	$d \leftarrow d+L0$	$\overleftarrow{M9}$	$P1-VL1+L2+L1+L0$	$V0$
6.	$V0$	$d \leftarrow d-P0$	-	$P1-VL1+L2+L1+L0-P0$	-

Table 4.2: Distance accumulation in the response of MP sequence (2) in Example 4.1

Remark The design of MP (and distance accumulation) supports a good level of *modularity* and *composability* of the model components in LIBROS. We deem the design modular in the sense that each (vehicle or RIE) model only uses its local information (viz., the state variables of the model component itself) and the information contained in the messages it receives to determine its own behavior. The direct dependency among the (vehicle and RIE) models is limited to the *couplings* designated for MP and to the *messages* being propagated.

Simple MP rules are defined (see § B.1) for cohesive model communications so that messages can be created, understood, manipulated and routed by the communicating parties. The models in LIBROS can be composed to work together in a modular manner without prior knowledge of the infrastructure model layout (which is of importance for the AMG presented in § 5) because of each model's observance of the communication protocol (or rules). The design of the model components and their inner working are presented from § 4.3.2 to § 4.3.5.

Note that MP is performed in LIBROS with zero time advance. This means when a vehicle model sends out a request message, it receives at least one and at most two response messages in zero simulation time.

4.3.2 An Overview on Infrastructure Models

The infrastructure models in LIBROS are characterized at a high level by whether the model is atomic or coupled and whether it has only one inflow and outflow of traffic or more. The former is related to the modeling formalism, and the latter to the domain specificity. Such an orthogonal abstraction can be designed as multiple inheritance and is designed as *mixin* in LIBROS (see, e.g., GAMMA et al. 1994, VIEGA et al. 1998).

4.3.2.1 High Level Infrastructure Model Classes

As shown in Figure 4.5 (note that the classes are abstract), we define the RIE models as `InfraElement` and the composed infrastructure models as `CoupledInfraModel` firstly by the respective specialization of the `AtomicModel` and `CoupledModel` classes provided by the ESDEVS library (see § A.2). Both infrastructure classes also implement `InfraInterface` which defines a number of common functions mainly concerning operations on the model ports and couplings; details see § 4.3.5.1.

The `CoupledInfraModel` class in LIBROS is used to define domain meta-models. It has two specializations. The *root* infrastructure model is declared as a (singleton) `TopLevelModel` with no parent. It serves as a container for the simulation model to be constructed (manual or automated) interfacing the simulator with the specification of the inner working of the model. It is the final outcome of the AMG described by § 5.

The second specialization of `CoupledInfraModel` is `InfraComponent` which is for defining meta-models of infrastructure compositions that are meant to be *component units* for model construction. By component units, we mean that the sub-components in an `InfraComponent` (as an unit) are strongly related such that their composition often reoccurs and can be reused as an assembled whole. An `InfraComponent` is defined

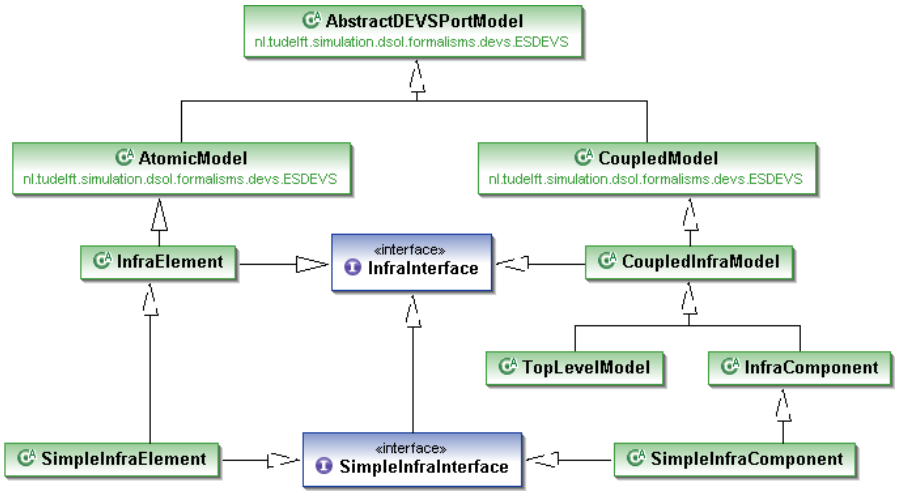


Figure 4.5: High level infrastructure model classes in LIBROS

for the convenience of model assemblance and reuse. We shall show in § 5 that `InfraComponent` definitions are used to match the graph patterns for AMG.

The `InfraElement` and `InfraComponent` classes each has a specialization (`SimpleInfraElement` and `SimpleInfraComponent`) that implements `SimpleInfraInterface`. The prefix `Simple-` indicates that the component being addressed has only one inflow and outflow of traffic. In other words, the component is not a switch or has no switch as a sub-component.

Model Class	Description
`InfraElement`	The RIE components, i.e., the atomic infrastructure models.
`SimpleInfraElement`	An `InfraElement` that has only one inflow and outflow of traffic.
`CoupledInfraModel`	The composed infrastructure models.
`TopLevelModel`	The root infrastructure model, a `CoupledInfraModel` that has no parent.
`InfraComponent`	An infrastructure component (unit) that is not atomic.
`SimpleInfraComponent`	An `InfraComponent` that has only one inflow and outflow of traffic.

Table 4.3: Infrastructure model class description

The reason we define `InfraInterface` and `SimpleInfraInterface` is twofold. On the one hand, we need to specify the models adhering to the chosen formalism whose template is provided by the chosen simulator. This naturally suggests a strong (specialization) relation of the models to the provided template as defined by the `AtomicModel` and `CoupledModel` classes (see § A.2). On the other hand, in order to refine the formalism outlined model behavior to meet the modeling needs of a domain, we need to add or "mix-in" more specific behavior that can be shared by some models. This can be achieved, e.g., by means of an interface.

4.3.2.2 A Design Pattern for Shared Alterable Mixin Behavior

A model wishes to have behavior refinement (or extension) may extend and implement a base `Model` and a `MixinBehavior` interface respectively, as shown in Figure 4.6. The interface allows for different versions of concretization of the declared behavior through *delegation* (see, e.g., GAMMA et al. 1994, VIEGA et al. 1998). More specifically, a `ModelWithMixinBehavior` acts as a *delegator* who delegates the mixin behavior to its *delegate* of type `MixinBehavior`. The latter carries out the concrete behavior. In Figure 4.6, two versions of mixin behavior (`Behavior1` and `Behavior2`) are shown. The `MixinBehavior`, if so wished, can be altered from a behavior concretization to another (cf. the *Adapter*, *Strategy* and *Bridge* patterns in GAMMA et al. 1994).

The *Shared Alterable Mixin* (SAM) pattern is *type safe* in the sense that both the delegator and the delegate(s) must conform to the set of behavior signatures declared by the `MixinBehavior` interface. A model has the *flexibility* of having more than one interfaces each of which can have multiple delegates. At the same time, different models can have delegates of the same interface type, affirming that delegation is a way of making composition[36] as powerful for reuse as inheritance (ibid.).

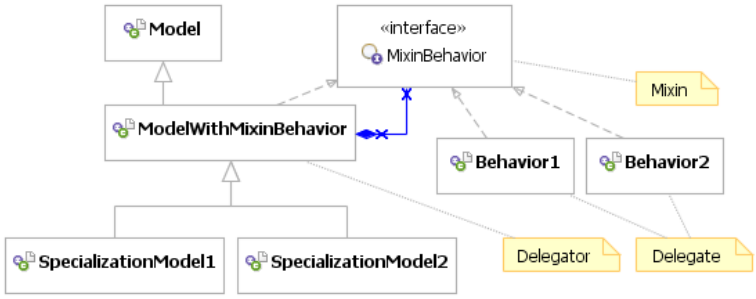


Figure 4.6: SAM pattern – a design pattern of components with shared alterable mixin behavior

[36] It refers to the object composition of strong "has a" not the model composition.

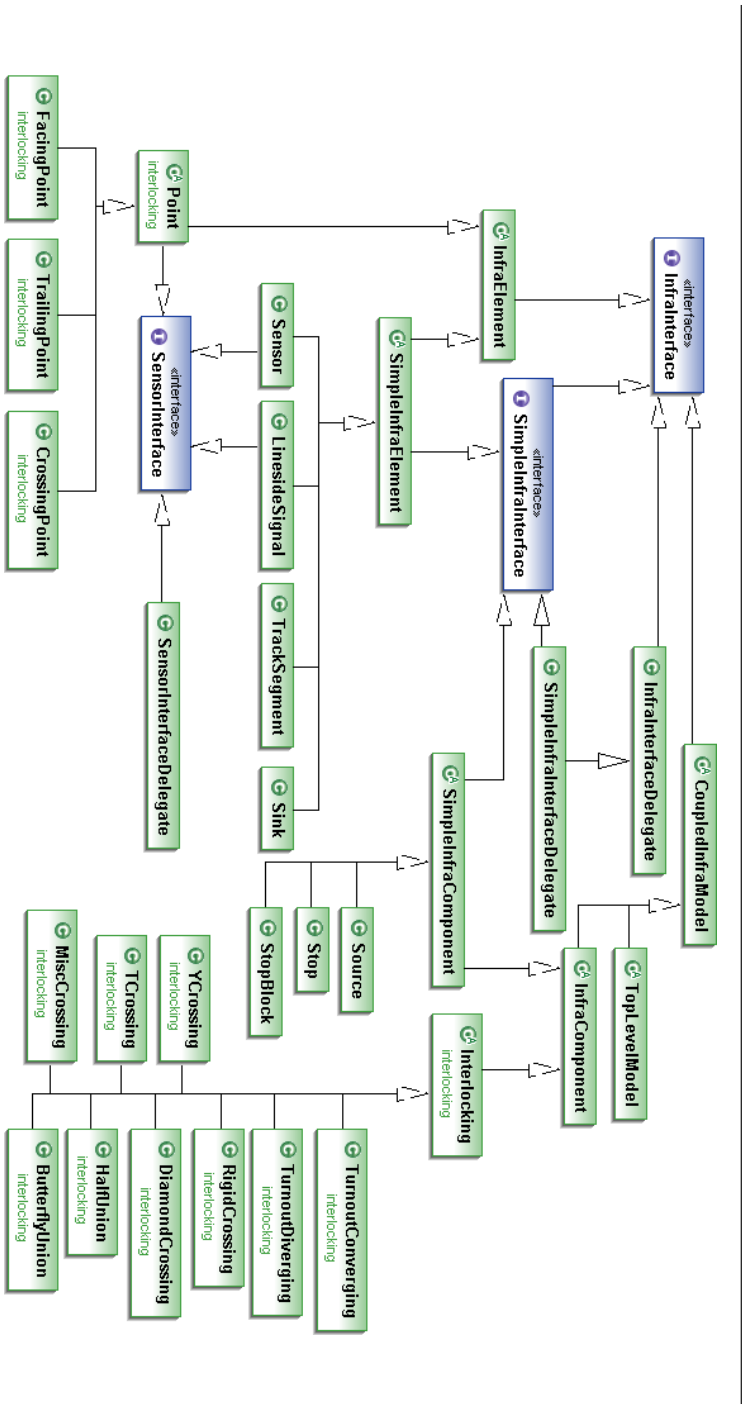


Figure 4.7: Infrastructure model classes in LIBROS

In the design of infrastructure models in LIBROS, the SAM pattern is used at three places. Figure 4.7 gives an overview of the design. Three mixin behavior interfaces are declared: `InfraInterface`, `SimpleInfraInterface` and `SensorInterface`. The first two mainly concern the operations on model ports and couplings (§ 5.3.1.4) that all infrastructure models in LIBROS would need. The third interface declares the operations that trigger and release some RIE models that also function as sensors (§ 4.3.4.4). The details about the infrastructure models are presented in § 4.3.4 and § 4.3.5.

Remark The SAM pattern can be used when a number of (atomic) model components need to have behavior or function composition, reuse and extension at the elementary level.

4.3.3 Vehicle Model

We prefer a discrete event representation of vehicle movement over a continuous or discrete time representation. The reason is stated in § 4.2.1[37]. Classic discrete event simulation models represent systems as piecewise constant state trajectory segments over a continuous time base (§ 2.4). This often does not produce satisfactory results for continuous and hybrid systems. In M&S literature, there are two approaches that permit modeling of continuous and hybrid systems based on the discrete event paradigm and the DEVS formalism in particular (KOFMAN et al. 2001, WAINER 2009): the Quantized DEVS (QDEVS) and the Generalized DEVS (GDEVS).

4.3.3.1 QDEVS vs. GDEVS

Quantization is briefly discussed in § 2.4.2. It discretizes (or quantizes) the state space rather than the time base as in *discretization*. The QDEVS (ZEIGLER 1998) provides a DEVS theory of quantized systems in which the space of state variables are quantized using an usually fixed value called the *quantum* (WAINER 2009). A state change occurs when the state variable crosses the threshold defined by the quantum (ibid.). The principle of conversion is similar to that of an analog-to-digital converter.

The Quantized State System (QSS) combines quantization with *hysteresis* (KOFMAN 2003a,b). Hysteresis is a technique that defines "buffer zones" (i.e., the hysteresis windows) around quantum thresholds which eliminate (possible infinitely) fast oscillations when a state variable "wanders around" a threshold.

The quantization approach is an alternative to the more traditional discretization approach. Both are able to simulate systems expressed in ordinary differential equations (ODEs) or differential algebraic equations (DAEs).

GDEVS (GIAMBIASI et al. 2000), on the other hand, is an approach that supports discrete event modeling of continuous and hybrid systems (WAINER 2009). It is, however, more general than classic DEVS in the sense that the state trajectory segments can be

[37] See § 4.2.1, fn. 19, p. 87.

polynomial instead of constant. In GDEVS, systems are represented as piecewise polynomial state trajectory segments over a continuous time base, in which the coefficients of the polynomial segments are piecewise constant (more see § 4.3.3.2).

Note 5 *The choice between QDEVS (or QSS) and GDEVS is in principle a choice between continuous and discrete event modeling styles.*

We choose GDEVS for vehicle movement modeling. Vehicle movement can be conveniently organized through piecewise polynomial segments where state changes occur when a vehicle changes its acceleration. The events that correspond with such state changes are well understood and can be rigorously defined.

Remark Modelers may prefer one style over another based on the nature of the real system and their personal preferences. When some systems can be more conveniently expressed in differential equations, QDEVS would be a good choice. GDEVS allows modelers to decompose a modeling problem into less complex ones and express them in polynomials which otherwise have to be expressed as a whole in differential equations. This is an advantage both for modeling and for model understandability. In order to model in GDVS, modelers have to know how a system responds to input events which are deemed significant. This can be seen as a disadvantage as the information is not always available or complete (WAINER 2009).

4.3.3.2 More On GDEVS

Following GIAMBIASI et al. (2000), GIAMBIASI and CARMONA (2006), a piecewise polynomial trajectory $w_{\langle t_0,t_m\rangle} \rightarrow A \subset \mathbb{R}$ is a collection of individual segments over a continuous time base with the following characteristics:

1. There exists a finite number of elements of instant $\{t_1, t_2, \cdots, t_{m-1}\}$ in the time interval $\langle t_0, t_m\rangle$. Each element in $\{t_i \in \mathbb{R}^+ \mid i \in [1, m-1] \subset \mathbb{N}\}$ is associated with a constant valued $(n+1)$-tuple $(a_{0i}, \cdots, a_{ni})$, $k \in [0, n] \subset \mathbb{N}_0 \wedge a_{ki} \in \mathbb{R}$, such that

$$\forall t \in \langle t_i, t_j\rangle, t_i \leq t_j, \text{where } t_i, t_j \in \{t_1, t_2, \cdots, t_{m-1}\} : w(t) = \sum_{k=0}^{n} a_{ki} t^k \tag{4.1}$$

   where n is the degree of the polynomial.

2. The trajectory satisfies the composition property (cf. Eq. 2.2 in § 2.1.1.1, p. 13)

$$w_{\langle t_0,t_m\rangle} = w_{\langle t_0,t_1\rangle} \bullet w_{\langle t_1,t_2\rangle} \bullet \cdots \bullet w_{\langle t_{m-1},t_m\rangle} \tag{4.2}$$

   where $\bullet$ is the left concatenation operator over the segments.

An individual polynomial segment $w_{\langle t_i,t_j\rangle}$ over the time interval $\langle t_i, t_j\rangle$ has coefficients $E(t) = (a_{0i}, \cdots, a_{ni})$ at a certain instant $t \in \langle t_i, t_j\rangle$, where the origin of time, i.e., $t = 0$, is assumed to occur at the start of each segment. This means that a piecewise polynomial trajectory can be represented through piecewise constant coefficients. As such, an event in GDEVS can be considered as a coefficient-event that activates or potentially activates a change of at least one of the coefficients (ibid.).

4.3.3.3 Modeling Vehicle Movement

We apply equations of motion in polynomials with constant acceleration since the simulation studies do not require kinematics or tractive force[38] considerations. Given a vehicle's velocity v_i and its assumed constant acceleration a_i at time t_i, one can calculate the vehicle's velocity v_j and movement distance s_i after a time interval $\langle t_i, t_j \rangle$ (where $t_i \leqslant t_j$) as follows

$$v_j = a_i \Delta t + v_i \; ; \quad s_i = \left(\frac{v_i + v_j}{2} \right) \Delta t = v_i \Delta t + \frac{a_i \Delta t^2}{2} \tag{4.3}$$

where $\Delta t = t_j - t_i$. In the two polynomials, Δt or simply t is the variable, and v_i and a_i are the coefficients that are constant in each polynomial segment.

Note 6 *Vehicle movement can be represented by piecewise polynomial segments each of which has a constant acceleration rate as a coefficient.*

When a vehicle completed a certain movement of a time interval $\langle t_i, t_j \rangle$, the value of its v_j, i.e., the new v_i value for the next polynomial segment, is determined. It still has to decide upon the new a_i value for the next segment as well as the anticipated time (let us call it $\Delta t'$ or t') of the next segment. In order to do so, a vehicle sends a message to request for information about its NCRI and PV, and it shall receive response messages in zero simulation time, as shown in Figure 4.8. The information request and response are accomplished by MP which is discussed in § 4.2.3 and § 4.3.1. In the following, we explain how a vehicle model uses the NCRI and PV information at hand[39] to compute a_i and t'.

A vehicle model can compute its movement to a position so far forth as safe driving is assured. Ideally, it *successively* computes the movement to each NCRI that it approaches.

Example 4.2 *Figure 4.9 illustrates an example of successive movement to the NCRIs. At time t_i position p_i, vehicle V has velocity v_i and acceleration a_i. Through MP, V knows that its NCRI is at distance d_{NCRI} position p_j with a speed limit l_{NCRI}.*

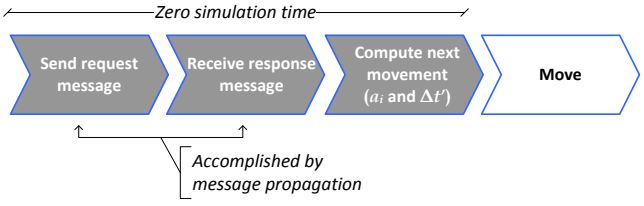


Figure 4.8: The action steps in one round of movement with message request

[38] The relevant formulas can be found, e.g., in HANSEN and PACHL (2008) pp. 61~80.

[39] The information includes a vehicle's distance to its NCRI d_{NCRI}, the speed limit or restriction l_{NCRI} required by the NCRI, and if applicable, a vehicle's distance to its PV d_{PV}, and the velocity v_{PV} and acceleration a_{PV} of the PV.

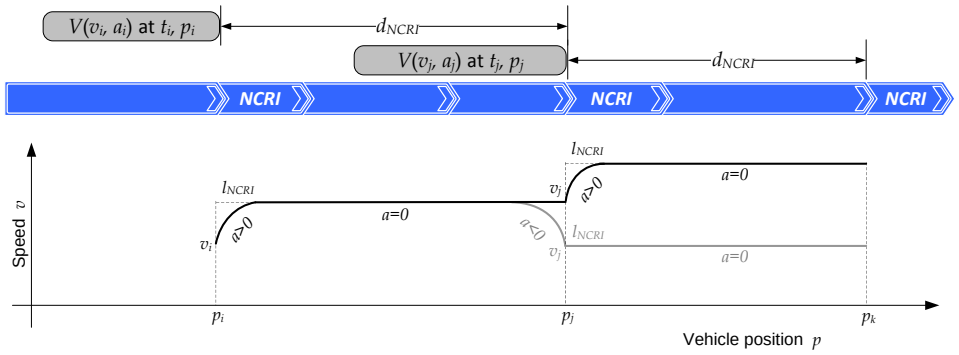


Figure 4.9: Vehicle movement computation with NCRI – Example 4.2

The vehicle can therefore compute its movement until reaching the NCRI at position p_j, say at time t_j. Because the value of v_i, the current speed limit l (i.e., the previous l_{NCRI}), and the current l_{NCRI} are not necessarily the same, the movement until the NCRI (from p_i to p_j in the example) can consist of *more than one polynomial segment* in which the value of acceleration a changes. In Figure 4.9, two polynomial trajectories of movement from p_i to p_j are shown in the speed-distance diagram for two different cases:

1. Suppose that $v_i < l < l_{\text{NCRI}}$, the movement would consist of two polynomial segments, viz., (1) $a > 0$, (2) $a = 0$.
2. Suppose that $v_i < l > l_{\text{NCRI}}$, the movement would consist of three polynomial segments, viz., (1) $a > 0$, (2) $a = 0$, (3) $a < 0$.

In the first case, a changes one time from a positive value to zero so that v remains constant afterwards until p_j. In the second case (shown in gray), a changes once more and becomes negative with which v reduces to l_{NCRI} at the moment when V reaches p_j. Once V is at p_j time t_j, the vehicle is dynamically coupled to the current NCRI instead of the previous NCRI to represent its changing of position. This starts a next round of MP and movement, and so iteratively in a similar manner until the vehicle reaches its final destination.

Example 4.3 *Suppose that the vehicle model in Example 4.2 has a PV in addition at distance d_{PV} at time t_i, as shown in Figure 4.10. The PV is not moving, i.e., $v_{PVi} = a_{PVi} = 0$, and its position is closer than that of the NCRI, i.e., $d_{PV} < d_{NCRI}$.*

When a vehicle model has a PV in addition, the computable movement to the NCRI as described in Example 4.2 *may* need to be shortened. This means that the distance d_{NCRI} may have to be divided into parts, each of which is to be computed separately. As

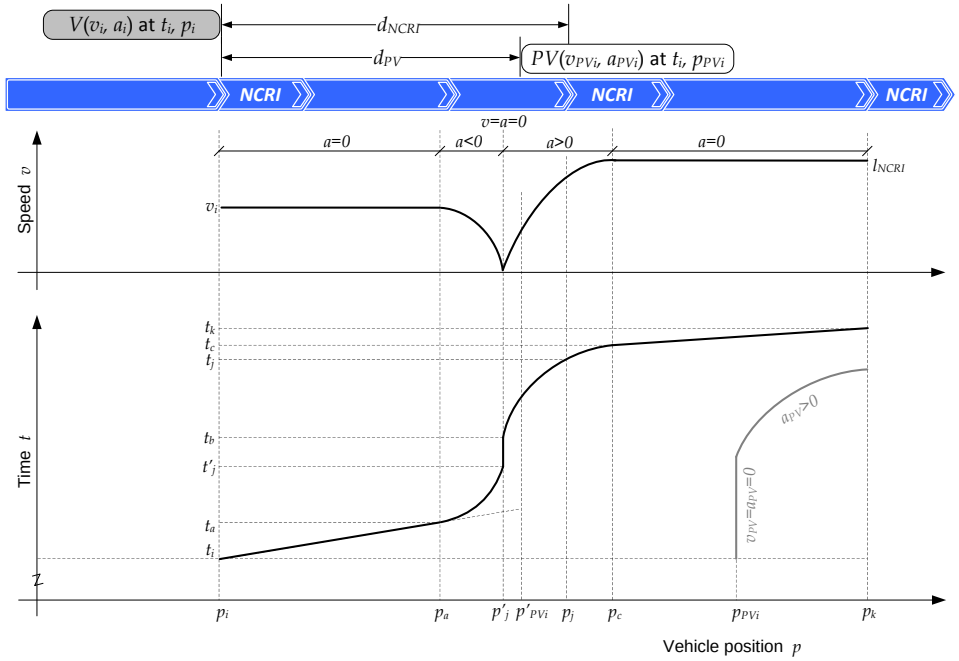


Figure 4.10: Vehicle movement computation with NCRI and PV – Example 4.3

common sense dictates, how V can move towards its NCRI is dependent on how its PV moves if the latter is close enough. Because the PV in the example is not moving and it is closer than the NCRI, V has to stop behind the PV. In other words, at time t_i, V can only compute its movement no further than the PV's rear end position $p'_{\mathrm{PV}i}$. The stop position is with some safety distance to $p'_{\mathrm{PV}i}$, shown as p'_j. During the movement, V must stop in time: it starts to brake at p_a time t_a, and after a time interval $\langle t_a, t'_j \rangle$ it stops at p'_j. Hence, the movement from p_i to p'_j consists of two segments:

(1) in interval $\langle t_i, t_a \rangle$, V moves with a constant speed v_i ($a = 0$) from p_i to p_a;

(2) in interval $\langle t_a, t'_j \rangle$, V moves with a decreasing speed ($a < 0$) from p_a to p'_j where its end speed reaches zero.

When V can drive again, its further movement is computed according to the new situation. This also means that V still needs to compute its movement to the NCRI. Suppose that some time after t'_j, the PV starts driving and it drives fast enough so that V can drive freely without considering the PV. In this situation, let us say that V can start driving at t_b. (We will soon come to the matter of how to obtain the value of t_b.) In principle, V can accelerate ($a > 0$) until the speed restriction l_{NCRI}. If so, the acceler-

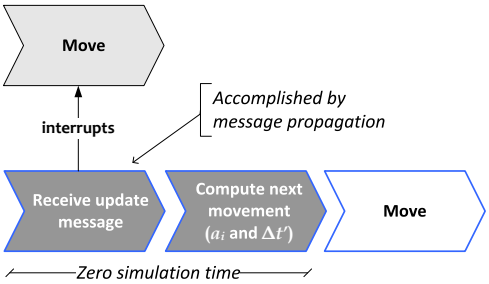


Figure 4.11: The action steps in one round of movement with an update message

ation would last for a time interval $\langle t_b, t_c \rangle$, starting at position p'_j ending at p_c. Suppose that the position of NCRI p_j is closer than the acceleration distance. The movement can only be computed until p_j where V has to be coupled to the NCRI at the time when V reaches it. This movement therefore consists of one segment:

(1) in interval $\langle t_b, t_j \rangle$, V moves with a increasing speed ($a > 0$) from p'_j to p_j.

Once V arrives at p_j, it is dynamically coupled to the NCRI, and this starts a next round of MP and movement.

With the above two examples, we try to elaborate how a vehicle model can successively compute its movement (which can consist of a number of piecewise polynomial segments) to its NCRIs in a discrete event manner taking into account the movement of its PV. In essence, a vehicle model computes the (internal) events of the following two groups of actions:

(i) when it shall move from one NCRI to the next, and

(ii) when it shall change its acceleration rate.

The actions are autonomous in the sense that a vehicle model decides locally its appropriate actions according to the information at hand. Every state change is handled by the model's internal transitions (§ 2.4.3). As explained in § 4.3.1, when a vehicle model changes acceleration, it sends out an update message (backward) so that a following vehicle, if any, can be informed of this change. Consequently, the (external) events of (the arrival of) update messages can be expected at any simulation time. This means that the execution of an anticipated movement may be interrupted by an update message from a PV, say at time t_{ext}. If so, the movement after t_{ext} has to be computed anew, as illustrated in Figure 4.11. In this context, "movement" or "move" is used in a general sense: it not only refers to a vehicle's moving actions but also non-moving actions such as scheduled or unscheduled waiting[40].

[40] An unscheduled waiting has an undefined waiting time which can be interrupted, while a scheduled waiting has a minimum waiting time which can not be interrupted.

Referring back to the waiting between $\langle t'_j, t_b \rangle$ in Example 4.3 (Figure 4.10), from time t'_j onwards, vehicle V is in an unscheduled waiting because its PV is on a halt. In this case, V is in passivity (or a passive state) meaning that it will remain idle unless an external event interrupts. When the PV starts driving some time after t'_j, V receives an update message from the PV of this event, which activates V to compute its movement according to the new situation. Time t_b is V's decision when it restarts driving. It can, e.g., wait until the PV has driven some distance.

The example shows that when an external event arrives, it interrupts and can diverge the anticipated state variable trajectories. Such interrupts often occur in the simulation of discrete event models and need to be treated with particular care when using piecewise polynomial abstractions.

4.3.3.4 Handling External Events

When a state variable is expressed by polynomial segments, the variable is not meant to be constant within a segment despite that each segment has constant coefficients. Its values are only computed[41] for chosen (external and internal) events. When an external event (or interrupt) arrives at time t_{ext}, a state variable still holds an old value that was updated at the past event. Thus, the value must be updated for the present time t_{ext}.

Note 7 *In GDEVS, a state variable shall be updated after the arrival of an external event before any state transition.*

Example 4.4 *Figure 4.12 depicts an example of state variable update in the vehicle model. The two state variables being updated are (1) the speed v and (2) the position p. An anticipated movement is computed from position p_i to p_k. It supposedly takes place in time $\langle t_i, t'_k \rangle$ and is consisted of two segments shown respectively with black lines in the speed-distance and time-distance diagrams:*

(1) in interval $\langle t_i, t'_j \rangle$, V moves with an increasing speed ($a > 0$) from p_i to p'_j in which the initial speed is v_i and the final speed is v'_j;

(2) in interval $\langle t'_j, t'_k \rangle$, V moves with a constant speed v'_j ($a = 0$) from p'_j to p_k.

Suppose that, during the first anticipated segment of $\langle t_i, t'_j \rangle$, an external event with an update message arrives at time t_{ext}. It diverges the movement trajectory away, shown with gray lines, from the anticipated movement.

The value of acceleration a is a result of movement computation, so is time t except for the arrivals of external events. Note that only a number of chosen points (or values) on the polynomial trajectories are computed. They are at the start[42] and the end of a

[41] Each polynomial has the form of Eq. 4.1 and satisfies Eq. 4.2. Expressed discretely, the ith segment has the initial value of a_{0i} (i.e., the coefficient of t^0). After an interval $\langle t_i, t_j \rangle$, assuming within which the polynomial is unchanged, the state variable would have a value of $\sum_{k=0}^{n} a_{ki} t^k$, which will be the initial value of a_{0j} of the jth segment.

[42] The start point is computed by the previous movement computation.

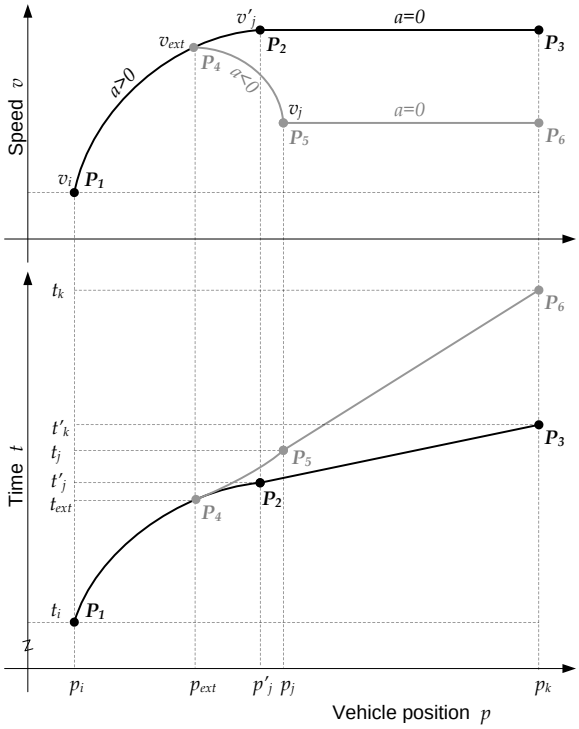
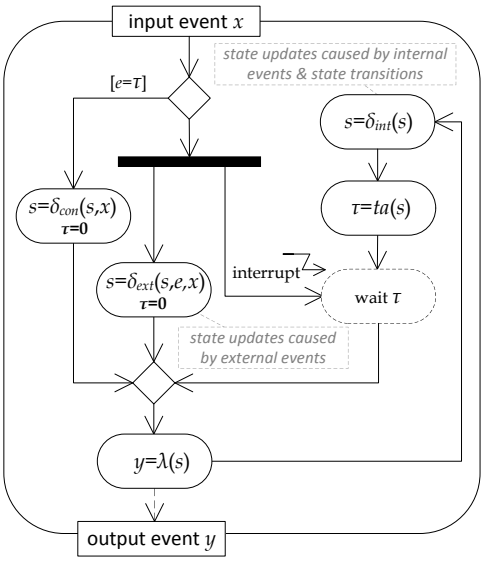


Figure 4.12: State variable update of speed v and position p in the vehicle model with an update message at t_{ext} – Example 4.4

trajectory, and where the polynomial segments change coefficients (i.e., only some dots not the lines). In the example, the values of three points are first computed. Shown in black dots (•) from left to right, they are $P_1 = (t_i, v_i, p_i)$, $P_2 = (t'_j, v'_j, p'_j)$, and $P_3 = (t'_k, v'_j, p_k)$ which pinpoint the trajectory of the anticipated movement $P_1 \xrightarrow{a>0} P_2 \xrightarrow{a=0} P_3$.

If no external event arrives during time $\langle t_i, t'_k \rangle$, the vehicle would "move according to the plan". In this case, two internal transitions are scheduled successively at time t'_j and time t'_k, by which the variables v and p are updated with the anticipated values of P_2 and P_3. Further movement is computed as explained in Examples 4.2 and 4.3.

In case an external event arrives during $\langle t_i, t'_k \rangle$, say at t_{ext}, the (anticipated) movement is interrupted and the vehicle's state variables must be first updated for the present time t_{ext} using the corresponding known polynomials for the time between $\langle t_i, t_{\text{ext}} \rangle$. If the event contains an update message for the vehicle, further movement after t_{ext} shall be computed anew; if the event contains a request message for the vehicle, it only needs to reply to the message.



►*internal transition first:*
$\delta_{con}(s,x) = \delta_{ext}(\delta_{int}(s), 0, x)$

Figure 4.13: SUTM – state update and transition mechanism of a DEVS atomic model in LIBROS

In Example 4.4, when the external event arrives, the state variables still hold the values of $P_1 = (t_i, v_i, p_i)$ which have to be updated to those of $P_4 = (t_{ext}, v_{ext}, p_{ext})$, the first gray dot (•) on the left. These values are subsequently used as the initial values for the new movement computation. Suppose that given the content of the update message, further movement is computed and we obtain discrete state variable points $P_5 = (t_j, v_j, p_j)$ and $P_6 = (t_k, v_j, p_k)$. These two points are used instead of P_2 and P_3 for the next two internal transitions. The new movement $P_4 \xrightarrow{a<0} P_5 \xrightarrow{a=0} P_6$ would take place in time $\langle t_{\text{ext}}, t_k \rangle$. It is as well an anticipated movement which may be interrupted by an external event as just discussed. If no external event arrives during this time, the total movement trajectory from position p_i to p_k would be $P_1 \xrightarrow{a>0} P_4 \xrightarrow{a<0} P_5 \xrightarrow{a=0} P_6$, in which the transition of P_4 is activated by an external event and the other three by internal events.

4.3.3.5 A State Update and Transition Mechanism

In § 2.4.3, we explained that an external event activates the external transition function $\delta_{ext}(s,e,x)$ in a model, which according to the definition in the DEVS formalism can change a model's state, i.e., $s' = \delta_{ext}(s,e,x)$. Consistent with the DEVS formalism, the external transition functions in LIBROS models do not handle the *state transitions* but only the *state updates* caused by the external events. Figure 4.13 illustrates how the DEVS atomic models in LIBROS handle state updates and transitions (cf. Figure 2.5 in § 2.4.3). Hereinafter, *state transitions* in a GDEVS model refer to the changes from one

polynomial segment to another; and *state updates* refer to the changes of state variable values within one polynomial segment.

Note 8 *The external transition functions in* LIBROS *models do not handle the "state transitions" but only the "state updates" caused by the external events.*

We also differentiate state updates caused by external events and internal events. The former is peculiar in GDEVS. An external event arrives at a non predetermined time (at least from the model's perspective) so that the values to be updated are computed instantaneously. In classic DEVS, as the state variables are constant between two successive events, such updates are not needed. State updates caused by internal events on the other hand are or can be predetermined as the internal events are determined by the model itself.

We reuse Example 4.4 to demonstrate the concept of handling state updates and transitions in LIBROS. Table 4.4 (p. 97) provides a list of the internal and external events that take place in the example and the corresponding state transitions and explanations. They are ordered by the movement trajectory $P_1 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6$ with the assumption that no external event takes place after time t_{ext} until t_k. The state update and transition mechanism (SUTM) can be summarized as following.

State update by δ_{ext} At the arrival of an external event, the δ_{ext} function updates the state variables (as it interrupts an internal transition), processes the message received, and if necessary it can have any other computation related to the external event. In a vehicle model, for example, the δ_{ext} may need to compute the anticipated movement with the latest information at hand. Note that the δ_{ext} only computes how the vehicle can move; the move itself is performed by the internal transition function δ_{int} not by the δ_{ext}.

State transition by δ_{int} The δ_{ext} in principle is *immediately* followed by the output function λ and the internal transition function δ_{int}. This means the time advance *ta* after the δ_{ext} is zero. The λ sends output events. In a vehicle model, they can be request, response and update messages. The δ_{int} handles all the state transitions as well as the state updates caused by the internal events[43]; if necessary it can have any other computation related to the internal event.

Benefits of SUTM The SUTM has a number of benefits. The state transitions of a model take place only at one location (the δ_{int}) instead of two (the δ_{ext} and δ_{int}), which improves *cohesion* in state transition logic when a model has strongly interrelated states. The λ directly follows the δ_{ext}, which is useful for *modeling interactions* among models. The fact that the δ_{ext} only handles issues related to the external events and leaves the state transitions to the δ_{int}, is a convenient way to model *autonomous* behaviors. This also makes the internal structure of an atomic model rather *simple* and the execution

[43] In case of a collision between external and internal events, the confluent state transition δ_{con} is configured to first execute the internal transition then the external which again sets τ to zero.

	Time	Event	Elapsed time e	Activated functions	Main actions	Time τ of next δ_{int}	Explanation
$\mathbf{P_1}$	t_i	Internal	–	λ, δ_{int}	↰ *Send update message* ∗ *State transition*	$t'_j - t_i$	Suppose the state is already updated. The δ_{int} would perform a state transition if the next acceleration is different from the previous one. If so, an update message is sent backward. According to the anticipated movement $P_1 \rightarrow P_2 \rightarrow P_3$, the next δ_{int} will be at t'_j.
$\mathbf{P_4}$	t_{ext}	External	$t_{\text{ext}} - t_i$	δ_{ext}	⟲ State update (from external) ⊟ Process update message ⊞ Compute movement	0	Upon an external event at $t_{\text{ext}} \in \langle t_i, t'_j \rangle$, the move from p_i to p'_j is interrupted and the planed δ_{int} is canceled. The δ_{ext} updates the state, i.e., calculates the values of v and p at P_4 using Eq. 4.3 where v_i, p_i, Δt (the elapsed time e), and a_i are known. The update message results in a new anticipated movement trajectory $P_4 \rightarrow P_5 \rightarrow P_6$.
		Internal	–	λ, δ_{int}	↰ Send update message ∗ State transition	$t_j - t_{\text{ext}}$	Immediately after the δ_{ext}, the λ and δ_{int} take place. State transition is performed because of the change of acceleration (from positive to negative). According to $P_4 \rightarrow P_5 \rightarrow P_6$, the next δ_{int} will be at t_j.
$\mathbf{P_5}$	t_j	Internal	–	λ, δ_{int}	↰ Send update message ⊛ State update & transition	$t_k - t_j$	The δ_{int} takes place as planed. It updates the state variables using the values of P_5. State transition is performed because of the change of acceleration (from negative to zero). According to $P_4 \rightarrow P_5 \rightarrow P_6$, the next δ_{int} will be at t_k.
$\mathbf{P_6}$	t_k	Internal	–	λ, δ_{int}	• Couple to the NCRI ↱ Send request message ⟲ State update (from internal)	∞	The δ_{int} takes place as planed. The anticipated movement is completed. Suppose the vehicle reaches the NCRI, it is coupled to the NCRI. The information for further movement is not at hand; thus the vehicles sends a request message forward. The state variables are updated using the values of P_6. Further movement is unknown so the vehicle model is passivated, i.e., $\tau = \infty$.
		External	0	δ_{ext}	⊟ Process response message ⊞ Compute movement	0	In zero simulation time, the vehicle receives a response message (or max. two) which results in a new anticipated movement.
		Internal	–	λ, δ_{int}	↰ *Send update message* ∗ *State transition*	n/a	Immediately after the δ_{ext}, the λ and δ_{int} take place. The δ_{int} would perform a state transition if the next acceleration is different from the previous one. If so, an update message is sent backward. The time of next δ_{int} depends on the newly anticipated movement. (Cf. the explanation of P_1.)

Table 4.4: The vehicle model state transitions in Example 4.4

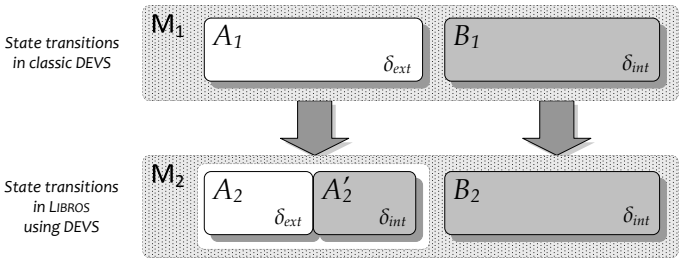


Figure 4.14: State transitions in a classic DEVS model vs. those in a LIBROS model

sequence of the state transition functions more *predictable* in the sense that the time advance *ta* after the δ_{ext} is always zero and the λ and δ_{int} are executed right after.

Costs of SUTM In a sense, the mechanism splits or separates one δ_{ext} into a δ_{ext} and a δ_{int} each of which is assigned with specific tasks. *Does this separation increase the number of scheduling of the* δ_{int} *in a model?*

Suppose that a model M_1 defined with classic DEVS has a set A_1 of transitions by the δ_{ext} and a set B_1 by the δ_{int} in a simulation run, as illustrated in Figure 4.14. With the same conditions, a comparable DEVS model M_2 defined with the mechanism discussed would have a set A_2 of transitions by its δ_{ext} and a set A'_2 by its δ_{int}, where the three sets related to external events have the same cardinality $|A_1| = |A_2| = |A'_2|$, i.e., the same number of transitions. That the δ_{int} in M_2 shall also function as the δ_{int} in M_1, would require a set B_2 by the δ_{int} in M_2, where $|B_1| = |B_2|$.

The extra number of internal transitions in M_2 compared to that in M_1 is therefore $\Delta n_{int} = n_{int,M_2} - n_{int,M_1} = |A'_2 \cup B_2| - |B_1| = (|A'_2| + |B_2| - |A'_2 \cap B_2|) - |B_1| = |A_1| - |A_1 \cap B_1|$ whose value depends on $|A_1|$ and how much A_1 and B_1 intersect[44]. This means that the more A_1 and B_1 intersect the less extra scheduling cost there would be, given that $|A_1|$ is fixed in a simulation run. This is the case when *a model actively responses to other models*, where after the δ_{ext}, the λ and δ_{int} are often directly activated.

Comparing the extra number of internal transitions with the total number of transitions in M_2, one gets the following:

$$\frac{\Delta n_{int}}{n_{total,M_2}} = \frac{\Delta n_{int}}{\Delta n_{int} + |A_1| + |B_1|} = \frac{|A_1| - |A_1 \cap B_1|}{|A_1| + |A_1 \cup B_1|}$$

This indicates that when $|B_1|$ is significantly larger than $|A_1|$, the extra cost for internal transitions compared to the total transition cost is insignificant. In many DEVS models, the number of internal events are greater than the number of external events, i.e., $|A_1| <$

[44] The intersection is at where the (transition) elements take place at the same simulation time, i.e., two transitions have the same time index.

$|B_1|$, because it is often the case that the interactions between the models are weaker than the interactions within them (the near-decomposability principle, SIMON 1996). There are of course cases when some models are designed such that it would have more external transitions, e.g., when a model is controlled by some other models or when a model is passive or idle for some purpose.

Remark Modelers should decide whether to use the SUTM based on the modeling context taking into account the benefits and the potential costs just discussed. We applied this mechanism often in the design of LIBROS models, as it suits the communication mechanism discussed in § 4.2.3 and § 4.3.1. We are in favor of simplicity and cohesion in the design of state transitions. They improve the understandability and potentially the extensibility of the model components. The SUTM is another example where we use the principle of *separation of concerns* in model design.

4.3.3.6 Rail Vehicle Model

A `RailVehicle` model has three ports as shown in Figure 4.15:

(1) `I/RIE` – an input port that receives messages from a RIE,

(2) `O/F/RIE` – an output port that sends messages forward to a RIE, and

(3) `O/B/RIE` – an output port that sends messages backward to a RIE.

All three ports are coupled to the RIE model where the vehicle is located. The vehicle model specification is a structure

$$\texttt{RailVehicle} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta) \tag{4.4}$$

where

$X = \{(p,m) | p \in \{\texttt{I/RIE}\}, m \in X_p\}$ is the set of *input port* and *messages*,
$Y = \{(p,m) | p \in \{\texttt{O/F/RIE}, \texttt{O/B/RIE}\}, m \in Y_p\}$ is the set of *output ports* and *messages*,
$S = S' \times S'' \times I_{NCRI} \times I_{PV}$ is the set of *states*, where
$S' = \{(\rho, a) | \rho \in \{\texttt{START}, \texttt{FOLLOW}, \texttt{MOVE\_TO\_NCRI}, \texttt{DWELL}, \texttt{STOP}\}, a \in \mathbb{R}\}$

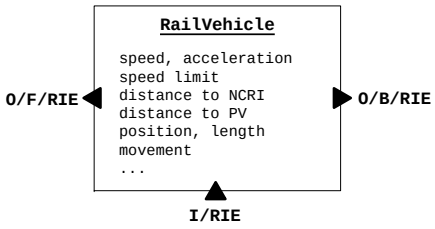


Figure 4.15: Rail vehicle model

	is the set of the vehicle's *primary states* which consist of its phase and acceleration,
$S'' = \{(v, \hat{v}, M, p, l, \cdots) \mid 0 \leq v \leq v_{max} \in \mathbb{R}^+, p \in \mathbb{R}_0^+\}$	is the set of *secondary states* which consist of its speed, speed limit, anticipated movement, position on the RIE where it is located, length, etc.,
$M = (M_0, \cdots, M_j, \cdots, M_k)$ where $M_j = (t_j, v_j, d_j, a_j), k \in \mathbb{N} \cup \varnothing, t_j \neq 0, a_{i \in [0,k-1]} \neq a_{i+1}$	is the *anticipated movement* consists of a number of successive movement segments each of which defined with the time (duration), end speed, moving distance, and acceleration of the segment,
$I_{NCRI} = \{(d_{NCRI}, l_{NCRI}, \cdots) \mid d_{NCRI}, v_{NCRI} \in \mathbb{R}_0^+\}$	is the set of *information about the NCRI* consist of the distance, speed limit, etc., of the NCRI,
$I_{PV} = \{(d_{PV}, v_{PV}, a_{PV})\}$	is the set of *information about the PV* consist of the distance of the PV and its speed and acceleration,
$\delta_{ext} : Q \times X^b \rightarrow S \backslash S'$	is the external transition function, where
$Q = \{(s, e) \mid s \in S, e \in [0, ta(s)]\}$	is the set of *total states* with elapsed time since last transition
$\delta_{int} : S \rightarrow S$	is the internal transition function
$\delta_{con} := \delta_{ext}(\delta_{int}(s))$	is the confluent transition function
$\lambda : S \rightarrow Y^b$	is the output function
$ta : S \rightarrow \mathbb{R}_0^+ \cup \infty$	is the time advance function

Note that a vehicle model's state is composed of its own state (primary and secondary) and the information it has at hand about its NCRI and PV. An external event only updates the S'', I_{NCRI} and I_{PV}, while the state transitions are performed by the internal events which change the S'. The details of the transition functions and how the vehicle model updates its state and computes its movement trajectories can be found in § B.2.

4.3.4 Rail Infrastructure Element Models

There are mainly four types of atomic Rail Infrastructure Element (RIE) models in LIBROS (cf. Figure 4.7, p. 86):

(1) tracks are of type `TrackSegment`,

(2) sensors are of type `Sensor`,

(3) signals are of type `LinesideSignal`, and

(4) switches[45] are of type `Point`.

[45] Switches are also called *points*, often by North American literature.

They all participate in MP (§ 4.2.3 and § 4.3.1). Because MP can be in forward and backward directions, for the convenience of model construction (and generation), we created type `Node`.

4.3.4.1 Nodes of Infrastructure Models

A `Node` is in principle a bundle of one input port and one output port dedicated for MP. It has as well a direction which indicates the inflow or outflow of traffic at an (atomic or coupled) infrastructure model, shown in Figure 4.16:

(1) a `StartNode` is a `Node` composed of one input port for message forward and one output port for message backward, and

(2) an `EndNode` is a `Node` composed of one output port for message forward and one input port for message backward.

When MP is in the direction of the traffic, it is in a forward direction, otherwise it is backward. The `StartNode` and `EndNode` are also used by coupled infrastructure models (§ 4.3.5.1). The port design of the RIE models is illustrated in Figure 4.17. The labels of the ports have the following meaning:

(1) `O/V` – an output port that sends messages to a vehicle model,

(2) `O/C` – an output port that sends messages to a control unit model,

(3) `I/C` – an input port that receives messages from a control unit model,

(4) `S` – a start node (at inflow of traffic) for MP at both directions, and

(5) `E` – an end node (at outflow of traffic) for MP at both directions.

From Figure 4.17, one can observe *cohesion* of the port design among the RIE models, as well as to the port design of the vehicle model (Figure 4.15). An `O/V` port is on each model (A)~(F). An `O/C` is on the 3S (sensor, signal and switch) models (B)~(F). In addition, a signal (C) or a facing point (D) has an `I/C` port because they are instructed by a control unit (§ 4.3.4.5). Each RIE model has at least one start node `S` and one end node `E`.

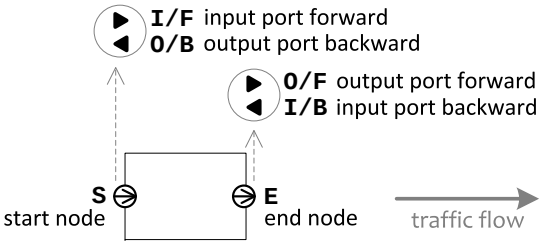


Figure 4.16: Start and end nodes of infrastructure models

The node-to-node coupling is a 1-to-1 relation. Note that the RIE-to-vehicle couplings allow 1-to-n relations. As such, when a RIE sends a message to a vehicle, the recipient is addressed. The same applies to the control-to-RIE couplings.

A RIE model has, among others, a length and a speed limit. (The 3S models are of length *zero*.) The role of `TrackSegments` in MP is already explained. The 3S models have similar roles in MP, but they always respond to request messages (except when the requester is already coupled to it) as they are at important positions. Compared to the `TrackSegments`, however, the 3S models have more interaction with the vehicle models. Besides MP, they can be triggered and released by a vehicle model (§ 4.3.4.3).

4.3.4.2 Vehicle Detection

The detection of occupation and clearance of tracks is fundamental to railway operation and control (PACHL 2002). In real life, different technologies are used for many types of detection devices (THEEG and VLASENKO 2009). To generalize the functionality

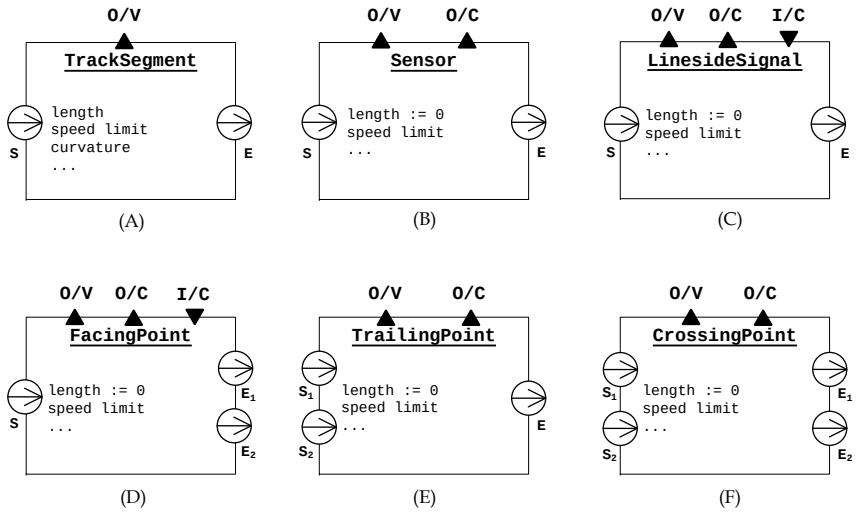


Figure 4.17: RIE models. With a facing point, the inflow traffic at `S` may exit at E_1 or E_2 depending on the position of the point which can be toggled by the control unit[a]. With a crossing point, the inflow traffic at S_1 only exits at E_2, and the inflow traffic at S_2 only exits at E_1. A crossing point models a location where two rail tracks cross each other.

[a] We did not define three-way switches. They can be modeled by two two-way switches.

and for simplification, the 3S models[46] are designed to have detection capacity (therefore they are also called *detector*s):

(1) the `Sensor` for detection,

(2) the `LinesideSignal` for detection and signaling, and

(3) the `Point`(s) for detection and switching (if applicable).

According to THEEG and VLASENKO (ibid.), three important detection purposes are:

(i) detect a vehicle reaching a certain point with its *front end*,

(ii) detect a vehicle passing a certain point with its *rear end*, and

(iii) detect track occupancy.

With the 3S models, purpose (i) is fulfilled by *triggering* a detector, purpose (ii) by *releasing* a detector, and purpose (iii) by two detectors at the two ends of a track. The 3S models can be coupled with the track segment models in different combinations to form different rail infrastructure layouts.

4.3.4.3 Trigger and Release of Detectors

When a vehicle model is approaching a detector, the latter must respond to the former's request message. The vehicle can then compute its movement until reaching the detector (which is its NCRI). Right after the movement, the vehicle's couplings to the former NCRI are replaced with the new couplings to the detector.

At this instant, the vehicle would initiate another request message in order to find the next NCRI. When the detector receives this message, it propagates it. Additionally the detector is triggered by the message and it shall compute the release (whose time is based on the vehicle's length, speed and acceleration) which is to take place at the next internal transition. Before the release time expires, if the vehicle changes its acceleration, it would send an update message to inform the succeeding vehicle. When the detector receives the message (because of the MP), it shall recompute the release time with the new information. If the detector receives other external events, it shall respond accordingly and then resume the release event. A detector's subsequent action upon the trigger or release is typically to send out a message to inform a control unit (§ 4.3.4.5). The detectors are used in combination with the control units to model railway operation and control rules.

4.3.4.4 Model Behavior Reuse with SAM Pattern

The 3S models have shared behavior. One can specify the behavior in each model individually, but a more convenient way is to *reuse* the behavior specifications. The SAM pattern (§ 4.3.2.2) is used for this purpose. Its application is illustrated in Figure 4.18 which is a subset of Figure 4.7 (p. 86).

[46] In real operation, the later two types do not have detection capacity per se, but there are detection devices placed around them.

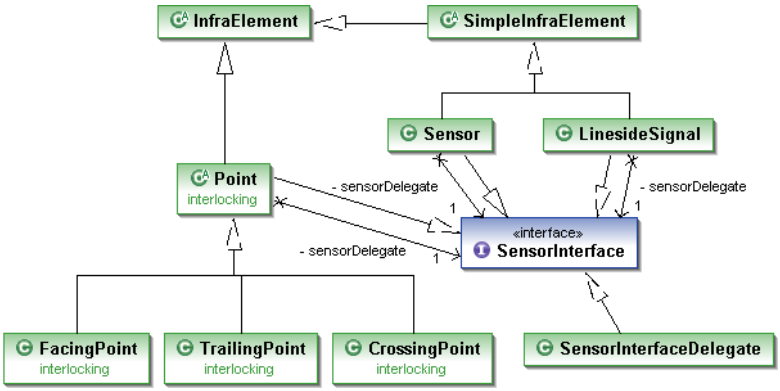


Figure 4.18: Sensor behavior reuse by the 3S models – a SAM pattern application

```
// base model:
abstract class InfraElement ... {
    // transitions:
    void deltaExternal(double e, Message m) { ...; }
    void deltaInternal() { ...; } // ...
}
// mixin behavior:
interface SensorInterface {
    void trigger();
    void release(); // ...
}
// delegate:
class SensorInterfaceDelegate implements SensorInterface {
    void trigger() { /*trigger action defined here*/ }
    void release() { /*release action defined here*/ } // ...
}
// delegator:
abstract class Point extends InfraElement implements SensorInterface {
    // has a delegate:
    SensorInterface sensorDelegate = new SensorInterfaceDelegate();
    // transitions:
    void deltaExternal(double e, Message m) {
        super.deltaExternal(e, m);
        // ...
        trigger(); // ...
    }
    void deltaInternal() {
        super.deltaInternal();
        // ...
        release(); // ...
    }
    // delegation:
    void trigger() { sensorDelegate.trigger(); }
    void release() { sensorDelegate.release(); } // ...
}
```

The `SensorInterface` is defined to specify the mixin behavior signatures that are shared by the `Sensor`, `LinesideSignal`, `Point`, and the `SensorInterfaceDelegate`. The former three are delegators of the detector (or sensor) behavior. Each has a delegate (of type `SensorInterfaceDelegate`) who indeed carries out the sensor actions, i.e., trigger and release, release time computation and update, among others.

The listing under Figure 4.18 shows the skeleton of SAM pattern implementation using `Point` as an example. The `Point` model (line 18) has `InfraElement` (line 2) as the base model and implements `SensorInterface` (line 8) to mix in the sensor behavior. It can reuse the transition functions (lines 4 and 5) of its base model in its own transition functions δ_{ext} (`deltaExternal`, line 22) and δ_{int} (`deltaInternal`, line 27) with which the trigger (line 25) and release (line 30) actions are also invoked respectively. These actions are however not performed by itself but delegated (lines 33 and 34) to its `sensorDelegate` (line 20) of type `SensorInterfaceDelegate` (line 13) where the actions are specified.

4.3.4.5 Control Unit

A *Control Unit* (CU) models the control logic of an area whose entrances are guarded by signals. As mentioned earlier, when detectors obtain information about vehicle positions, they send the information to CUs. The latter evaluates the information and permits vehicle movement via signals (THEEG and VLASENKO 2009).

The `ControlUnit` models in LIBROS are not coupled with `TrackSegments`; neither do they participate in MP. They receive input events only from detectors, and send (control) messages to `LinesideSignals` or `FacingPoints`. A `ControlUnit` model hence has two ports as shown, in Figure 4.19, *cohesive* to the ports of the RIE models:

(1) `I/S` – an input port that receives (sensor) messages from a sensor (i.e., detector),

(2) `O/LS/FP` – an output port that sends (control) messages to a lineside signal or a facing point.

A Crossing with Control Unit Suppose we need to model a simple tram crossing whose layout is illustrated in Figure 4.20 (A). The layout is similar to that in Example 3.1 (Figure 3.2); some may prefer to regard them as the same. Let us call this kind of crossings "Y crossing".

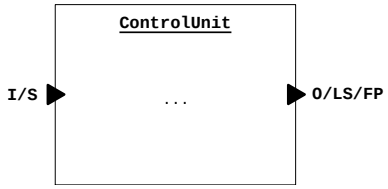


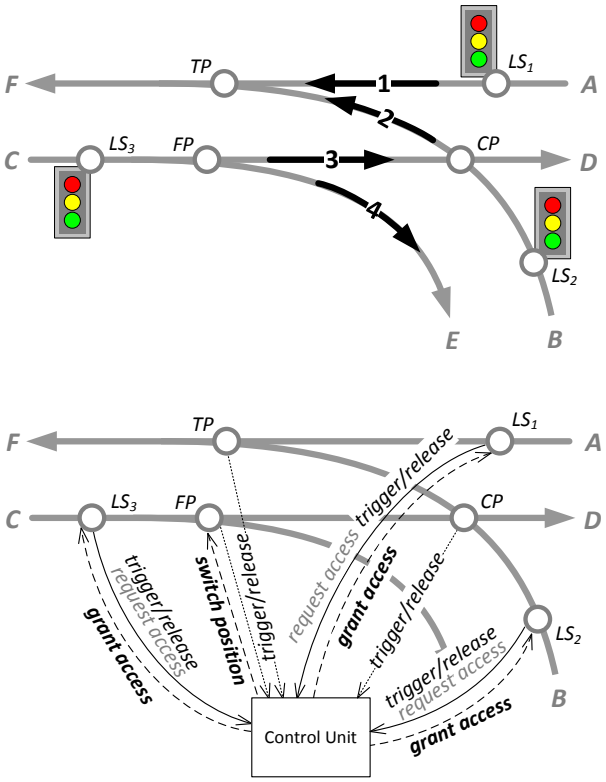
Figure 4.19: Control unit model

Example 4.5 *A Y crossing is guarded by three lineside signals ($LS_1 \sim LS_3$). There are three switches in the crossing: a trailing point (TP), a facing point (FP), and a crossing point (CP). Four directions (or routes) of traffic are possible:*

1. *from point A to point F ($A \rightarrow F$),*
2. *from point B to point F ($B \rightarrow F$),*
3. *from point C to point D ($C \rightarrow D$), and*
4. *from point C to point E ($C \rightarrow E$).*

The control logic is common sense, i.e., when a track or switch is occupied, it can not be accessed by others.

In a LIBROS model, a crossing is supervised by a CU which is responsible to grant permissions to enter the crossing. The couplings of the 3S models with the CU at the Y crossing are shown in Figure 4.20 (B). Note that the inputs or outputs to a CU are



(A) Layout

(B) Coupling with control unit

Figure 4.20: A Y crossing – Example 4.5

coupled with only one input or output port of the CU respectively. An input event from a 3S model to a CU contains information about sensor trigger or release. In addition, a signal can send a request to a CU on behalf of an approaching vehicle for the access of the crossing. An output event from a CU to a signal takes place when a requested access is granted. In addition, a CU can ask a facing point to change position (viz., left or right) when necessary.

Example 4.6 *Suppose that a vehicle V is approaching the Y crossing (as described in Example 4.5) at some distance and wants to drive from C to D, i.e., route 3, as shown in Figure 4.21.*

At some instant, V would send a request message (to look for its NCRI) which is received by the signal LS_3. Knowing V is approaching, LS_3 sends a message on behalf of V to request for the access of route 3. Upon the request, the CU checks if the route is accessible. The CU does so by consulting a "check table".

Check Table in Control Unit A *check table* in a CU in principle maintains the notes or records about the situation in an area the CU is supervising. For convenience, the CU keeps in the table the information about the routes in the area, the required points (and positions if applicable) by the routes, the states (whether reserved and/or occupied) of the points, weather the routes are active or have queuing requests, etc. An example of the table is shown in Table 4.5.

A route in a crossing is *active* when a request of the route is granted to a vehicle *and* the vehicle has entered or is about to enter the route but has not yet left the route (i.e., the requesting vehicle is at the crossing). If so, when another vehicle requests the same route, the request will be granted by the CU (by the default setting of the model). When a route has *queuing request*(s), a new request for the route will also be queued.

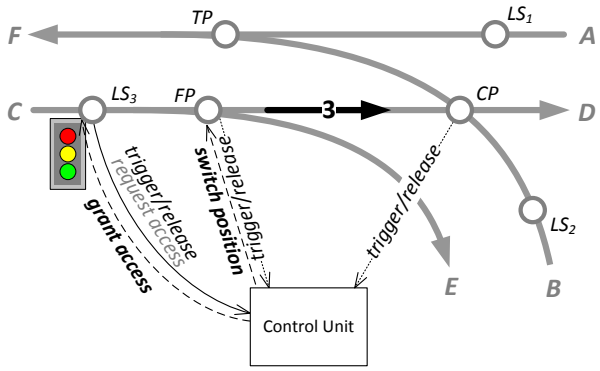


Figure 4.21: Access of a route in a Y crossing – Example 4.6

Route	Entrance Signal	Points required	Active	Queuing requests
1	LS_1	*TP*		
2	LS_2	*CP*, *TP*		
3	LS_3	*FP*/L, *CP*		
4	LS_3	*FP*/R		

Table 4.5: A check table in a control unit – Example 4.5, 4.6

Referring back to Example 4.6, when the CU receives the request to grant access to route 3, it processes the request by first checking whether the route is active or has queuing requests. Suppose both are negative, the CU then checks whether the points required by the route (i.e., FP and CP) are reserved or occupied. If they are, then the request will not be granted but is appended into the queuing requests of route 3. If not, then the request will be granted. In the latter case, the CU marks route 3 as active and the required points as reserved in its check table, and sends out a message to announce this decision. Note that the CU has one output port `O/LS/FP` that is coupled to $LS_1 \sim LS_3$ and the FP. Two recipients are addressed in this message:

(1) the FP whose position-to-be is noticed, and

(2) the LS_3 (i.e., the *entrance signal* of route 3) who will give the requesting vehicle a green signal (i.e., a message).

A vehicle who is permitted to enter a requested route will trigger and release the detectors along the route. The CU then marks the detectors as being occupied and unoccupied/unreserved respectively in its check table. Releasing a signal turns the signal back to red (which is its default state). When all detectors along the route are released, the CU marks the route as *inactive*. Once a point is released, the CU will also process the queuing requests (if any) that require this point. They are processed in an order of vehicle priorities (if any) and on a *first-come first-served* basis.

4.3.5 Coupled Infrastructure Models

The RIE (i.e., `InfraElement`) models discussed in § 4.3.4 are atomic. They can be used to incrementally compose *Coupled Infrastructure Model*s (CIMs), i.e., `CoupledInfraModel` in LIBROS. The CIMs can be used for composition as well. They are defined to represent domain meta-models that allow for a set of model compositions.

4.3.5.1 Node Couplings and Operations of Infrastructure Models

A CIM can be seen as a placeholder or a container for a set of infrastructure models. As shown in Figure 4.22, a `CoupledInfraModel` holds a list of coupling relations of its

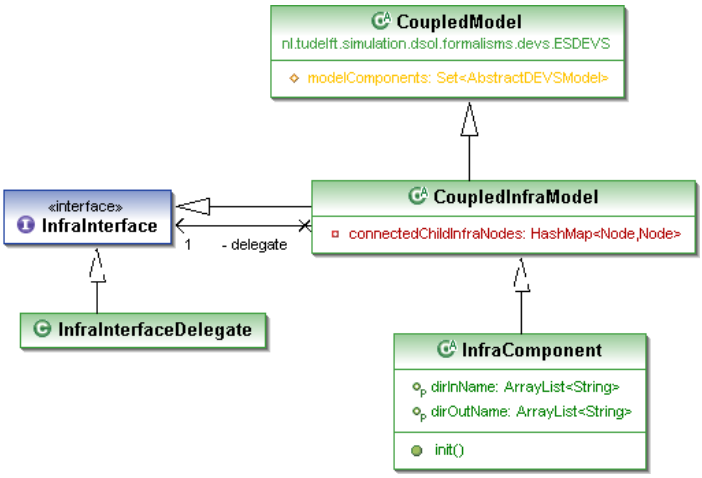


Figure 4.22: Coupled infrastructure models

infrastructure sub-components. The coupling relations are specified in terms of nodes (§ 4.3.4.1). Each node-coupling is comprised of two port-couplings. Figure 4.23 (cf. Figure 4.16) shows an example of a CIM M with two (infrastructure) sub-components M_1 and M_2. The rules for the EIC, EOC and IC couplings in DEVS models (§ 2.4.3.3) also apply to node couplings.

A node-to-node coupling relation is strictly 1-to-1. For example, the outflow node of an infrastructure model can only connect to one inflow node of another infrastructure model (which is not the former's parent model). An infrastructure layout that has only

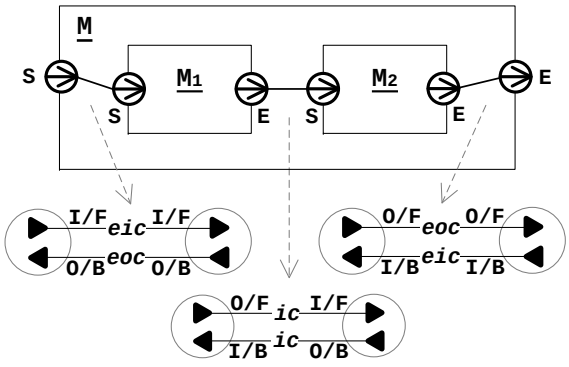
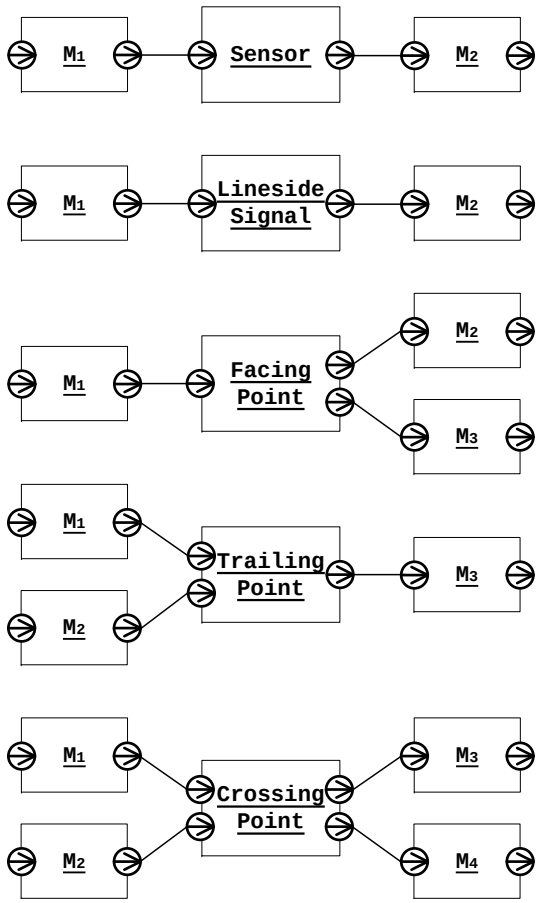


Figure 4.23: Couplings of nodes of infrastructure models

one inflow and one outflow of traffic is hence represented by a chain of successively connected infrastructure models each of which has only one inflow node and one outflow node. We call this type of (atomic or coupled) infrastructure models `SimpleInfra-` (§ 4.3.2.1). The models in Figure 4.23 are examples.

Note that the layout of "a sensor or a signal on (or along) a track" has to be represented by a chain of successive connections, similar to that shown in Figure 4.24 (A) and (B). Connecting, e.g., `M1` through one outflow node with `Sensor` (or `LinesideSignal`) and with `M2` at the same time will cause a *non deterministic execution sequence of the external transition functions* δ_{ext} in the two latter models when there is an output event at `M1`. Both δ_{ext} shall be and will be executed at the *same simulation time*, but which one is executed first is dependent on the simulator.



(A) Senor (1-to-1)

(B) Signal (1-to-1)

(C) Facing point (1-to-2)

(D) Trailing point (2-to-1)

(E) Crossing point (2-to-2)

Figure 4.24: 3S model coupling

Non determinism is in general undesirable in M&S. A non deterministic execution sequence of different external transition functions, which are supposedly to be executed at the same simulation time, may cause but does not necessarily result in non deterministic simulation results. If it is the case, measures have to be taken to make the execution sequence deterministic. Because this non deterministic execution sequence would have a negative effect on our simulation, a non 1-to-1 node-to-node coupling is defined as impermissible in LIBROS. In this context, a sequential connection is a measure that forces a sequential activation of the aforementioned external transition functions.

Note 9 *One-to-more DEVS port couplings should be applied with caution because they may cause a non deterministic execution sequence of the external transition functions activated by the same external event.*

Note 10 *A non one-to-one node-to-node coupling is impermissible in* LIBROS.

An infrastructure layout with more than 1-to-1 traffic is represented through the use of `Points`. Illustrated in Figure 4.24 (cf. Figure 4.17), points may have 1-to-2, 2-to-1, or 2-to-2 inflow and outflow of traffic (C)∼(E). Combinations of them can represent an n-to-n infrastructure layout.

A `CoupledInfraModel` needs a number of *functions* that can operate on the model nodes and couplings as well as on its sub-components. For the convenience of model construction (and generation), we defined operations to allow for, e.g., adding and removing sub-components, as well as adding, removing and managing nodes and the couplings of nodes dynamically at model generation time (§ 5.3). Since the node operations are also needed by the RIE models (i.e., `InfraElement`), the SAM pattern (§ 4.3.2.2 and § 4.3.4.4) is applied to reuse the definition of these operations among the infrastructure models. As shown in Figure 4.22, the `CoupledInfraModel` implements the `InfraInterface` (so does the `InfraElement`, cf. Figure 4.5) in order to delegate the operations.

4.3.5.2 Infrastructure Components

In § 4.3.2.1, we explained the difference between the `CoupledInfraModel` and the `InfraComponent`. While the former can be any CIM, the later is a CIM with a specific layout or setting which is meant to be reusable as a whole. An `InfraComponent` can be seen as a component unit, whose sub-components are strongly related in the sense that they have some common operation and/or need common control logic for the well functioning of the component unit.

For example, the Y crossing discussed in Example 4.5 (Figure 4.20) is defined as an `InfraComponent` (`YCrossing`; see Figure 4.7) first because it has a common infrastructure layout (pattern) and also because the 3S models in the crossing need to be supervised by the same control unit for the safe operation at the crossing.

An `InfraComponent` model in LIBROS is defined with possible compositions and constrains in such a way that it serves as a meta-model (§ 2.2.1). Each sub-class of

the `InfraComponent` model class (see Figure 4.7) defines a meta-model of a certain infrastructure composition whose (infrastructure) compositional feature is described by a corresponding graph pattern. The latter is used for graph pattern matching in the infrastructure data for the AMG presented in § 5.

In the AMG, a graph pattern is a *source meta-model* of a subgraph in an infrastructure data graph; an `InfraComponent` model is a *target meta-model* of a sub-component in a `TopLevelModel` (see § 3.4, Figure 3.6). An `InfraComponent` model also defines a set of operations on how to use a matched subgraph to instantiate (and configure) a corresponding model component. As this is closely related to the AMG, we will give examples of infrastructure components in § 5.

4.3.5.3 Vehicle Coupling

In § 4.2.3, § 4.3.1 and § 4.3.3.3, we discussed the design of vehicle communications and dynamic couplings. A vehicle model is generated (in a source model, see § 4.3.5.4) at a certain instant during simulation, and then dynamically connected to a sequence of NCRIs one at a time representing where the vehicle is located. When it reaches its destination, it is removed (by a sink model, see § 4.3.5.4) from the simulation.

Dynamic structure DEVS (DSDEVS) allows for the change of model structure at simulation run time. There are two popular definitions of DSDEVS in literature (WAINER 2009), one by BARROS (1997) and the other by UHRMACHER (2001). The latter is used by LIBROS as it is the definition that is implemented by the ESDEVS simulator (§ A.2).

Figure 4.25 gives an example of vehicle couplings. A vehicle model V is successively coupled with the RIE models *RIE*s at time intervals corresponding to the time that is needed by the vehicle moving from one NCRI to a next. It always has three couplings with a *RIE* at any simulation time:

(1) one coupling for input messages – from *RIE* output port `O/V` to V input port `I/RIE`,

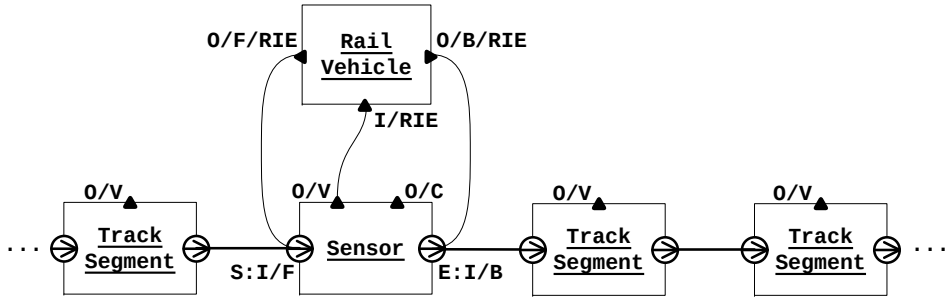


Figure 4.25: Rail vehicle model coupling with RIE models – an example

(2) one coupling for output messages forward – from *V* output port `O/F/RIE` to *RIE* start node `S` input port `I/F`, and

(3) one coupling for output messages backward – from *V* output port `O/B/RIE` to *RIE* end node `E` input port `I/B`.

All *V-RIE* couplings are designed in this manner since the RIE models[47] are designed to have the required ports. Therefore, the configuration function for the dynamic *V-RIE* couplings can have *cohesive* operations. When a vehicle model reaches its NCRI, it invokes by itself the configuration function COUPLETONCRI for its dynamic couplings, in which the couplings with the previous NCRI are removed and new couplings with the current NCRI are established. As discussed in § 4.3.3.3 and § 4.3.3.5, a vehicle model alwasys reaches its NCRI at an internal event. If so, the COUPLETONCRI function is invoked from the output function λ (see § B.2.2, line 10) after which a new round of MP is initiated.

4.3.5.4 Source and Sink

A LIBROS model contains at least one source and one sink where vehicle models are generated into and removed from the model during a simulation run. The source and sink models are of a special type of infrastructure model in the sense that "no traffic flows through" the model. This means that only one of the two nodes of a source or sink model is connected. Their design is shown in Figure 4.26.

A `Source` is a coupled model that contains one virtual track and at lease one (atomic) vehicle generator. One `VehicleGenerator` can generate vehicle models according to one timetable or one time interval distribution. Once generated, a vehicle is coupled to the `TrackSegment` in the source and assigned with a position closest possible to the end of the track. The vehicle model then sends a request message at initialization[48] after which it decides on its own movement (§ 4.3.1 and § 4.3.3). We use a track in a source model for two reasons:

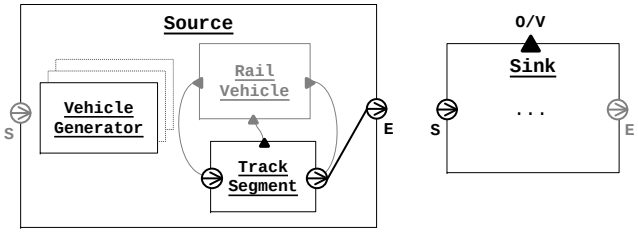


Figure 4.26: Source and sink models

[47] When a RIE model has more than one start or end node, a coupling with either will give correct results. The default setting is to connect with the first start or end node.

[48] The initialization refers to the very first internal event of the DEVS atomic model; see Figure A.3.

(1) when there are more than one vehicle in the source, the track is a means to queue the vehicles by the time they are created;

(2) the track is also a means that allows a vehicle to send out a request message and to drive out from inside the source, which is cohesive with the other infrastructure models.

To couple a newly generated vehicle directly with a track outside of a source, instead of a track inside, would break the modularity of the source model. A track inside a source model is virtual as it does not model any part of the "real" infrastructure and the mileage inside a source is not accounted for statistics.

A `Sink` responds to a request message in the same manner as the RIE models. But once a vehicle reaches a `Sink`, it is removed from the simulation: the vehicle's couplings are removed and the vehicle model is removed from the component list of the parent model. Because the vehicle's succeeding vehicle (if any) still holds the information about its PV, which would cause unconscionable movement computation of the latter, the `Sink` sends a backward message to inform the latter who then clears its PV information.

4.4 A Study on LIBROS Model Performance

In this section, we present a study that compares a LIBROS model with a model that uses continuous abstractions. The work reported by KANACILO and VERBRAECK (2005, 2006, 2007) also developed a rail simulation library in which vehicle movement is represented by differential equations. We hereinafter call this library LIBODE. We deem LIBODE and LIBROS comparable because they are developed with similar modeling details and purposes, and both libraries have the same underlying (event scheduling based) simulator which is called DSOL (JACOBS 2005). LIBODE is based directly on DSOL and LIBROS is based indirectly on DSOL through the ESDEVS library (SECK and VERBRAECK 2009). The dependency relation is depicted in Figure 4.27. More information about DSOL and ESDEVS can be found in § A.2.

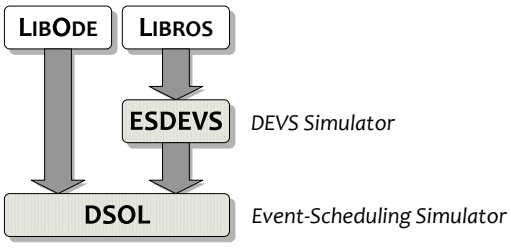


Figure 4.27: LIBODE and LIBROS

4.4.1 Experimental Setup

Using the LIBODE and LIBROS libraries respectively, we created two models representing the light-rail operation in The Hague city center tunnel in the Netherlands. This location is chosen for its simpleness but suffices to show examples of LIBODE and LIBROS models, and to compare them in terms of their performance.

4.4.1.1 LIBROS Model

The LIBROS model of the light-rail operation in the tunnel is shown in Figure 4.28. Only one direction (from left to right) of traffic is illustrated; the other direction is modeled similarly. In the tunnel, there is a stop with two halting places (T_3 and T_4). The stop is guarded by a signal (*LS*) which signals vehicles to wait (i.e., red) when either T_2 or the second halting place (T_3) is occupied by a vehicle. When the first halting place (T_4) is occupied and both T_2 and T_3 are not, the signal shows amber. The signal turns green when the block section is completely cleared. The speed limit in this tunnel is 45 kmph and it is reduced to 35 kmph when the signal is amber.

The MCT of the LIBROS tunnel model is illustrated in Figure 4.29. The four nodes in the MCT are `CoupledInfraModels`, among which the root is a `TopLevelModel`. The *Tunnel Model* is compose of a *Block Section*, two track segments T_1 and T_5, a *Source* and a *Sink*. The *Block Section* is composed of a lineside signal *LS*, a *Control Unit*, and other components as shown. Besides vehicle detection, the sensor S_1 in the *Stop* is also used by a vehicle to identify the position of the stop. Another sensor S_2 is placed at the end of the block section to detect clearance.

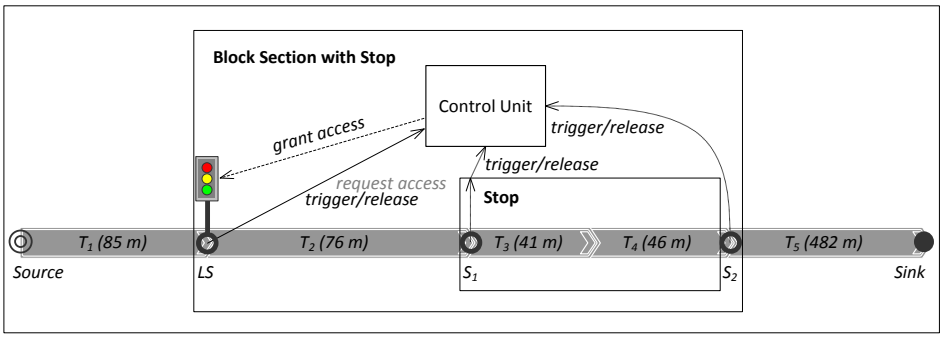


Figure 4.28: Light-rail operation in The Hague city center tunnel modeled with LIBROS

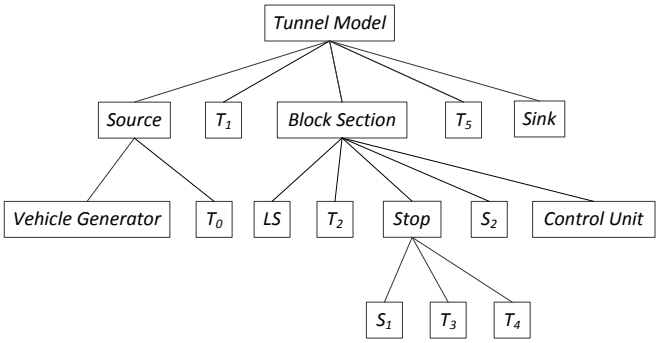


Figure 4.29: Composite tree of the LIBROS tunnel model

4

4.4.1.2 LIBODE Model

The same infrastructure and control logic are modeled with LIBODE. The LIBODE tunnel model has one track T that comprises the total length of $T_1 \sim T_5$ in the LIBROS model. The block section or the stop is defined with a position associated with the track; e.g., the block section is on T, offset 85 meters, length 163 meters. The positions of the signal and sensors are defined in the same manner.

In LIBODE, communications are mainly accomplished through *publish-subscribe*[49] which is implemented by DSOL. For example, when a vehicle is approaching a signal, the state changes of the signal would be of the interest to the vehicle. The vehicle model thus registers itself to the signal's *subscribers list* and becomes a listener of its state changes. Once the vehicle passes the signal, it unregisters itself from the list.

The vehicle movement is represented by differential equations solved by the Runge-Kutta integrator contained in DSOL. In principle, at each integration step, a vehicle model decides if it shall accelerate, cruise or brake based on the states of the objects (i.e., vehicles, signals, sensors, etc.) associated with the track(s) ahead of it. If a new object is in sight, the vehicle subscribes to the model so that it can be notified when the model changes states.

4.4.1.3 Experiments

For each model, we ran experiments with different vehicle generation frequencies (VGFs), viz., 15, 22, 30, 40, 50, 60 vehicles per hour (vph). Two measures are taken to ensure that some vehicles do interact with another.

[49] The **publish-subscribe** (or event notification) interaction mechanism defines an asynchronous non static one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated (GAMMA et al. 1994). More specifically, subscribers register their interests in an event (or an event pattern) with publishers, and are subsequently notified each time when such an event is generated by the publishers (EUGSTER et al. 2003).

(1) The vehicle generation interval is not uniformly distributed but with time variations. When, e.g., the VGF is 15 vph, generating one vehicle every four minutes would likely yield no interactions as the total driving time through the tunnel is short (∼1 min in the best case without halting) and there exists only one stop that can delay this time.

(2) The halting time at the stop is configured such that it is long enough (i.e., more than the average vehicle generation interval) to yield vehicle interactions. When, e.g., the VGF is 30 vph, the halting time would be more than two minutes.

Each pair of experiments is set with the same configurations for both models so that the results are comparable. The LIBODE model is simulated with an integration step of 1/20 second which in the worst case has a vehicle position accuracy of 0.625 meter, considering the speed limit of 45 kmph. The vehicle position in the LIBROS mode is computed as described in § 4.3.3.3 (with floating point precision). The simulation run length of each experiment is two hours. Each simulation run is profiled with the TPTP tool[50].

It is expected that the VGF and the computational cost of the simulation have a positive correlation, since more vehicles create more communications in both models, more need for integration in case of the LIBODE model, and more state transitions in case of the LIBROS model. The difference of degrees of this correlation in both models is what the experiments want to find out.

4.4.2 Experiment Results and Discussion

The execution time measurement of the LIBODE and LIBROS tunnel models is presented in Figure 4.30. In both cases, the results show that the model execution time increases nearly linearly with the increase of VGF. However, the execution time of the latter is consistently lower and increases slower compared to that of the former. Although the model execution time in the experiments is not large, performance becomes important for large-scale microscopic M&S.

We profiled the experiments with the VGFs of 15, 30 and 60 vph, and then categorized the *execution base time*[51] by the major functionality (with the highest base time) used by the two models. The results are shown in Figure 4.31.

As mentioned earlier, both the LIBODE and LIBROS libraries are based on DSOL. LIBODE uses DSOL for integration and model communications (i.e., publish-subscribe), which need the event-scheduler in DSOL. LIBROS uses ESDEVS for model state transitions and communications (i.e., port-based message passing); these events are scheduled by ESDEVS using the event-scheduler in DSOL as well.

Because the base time is taken as the time measure for the experiments, the time spent for integration by the LIBODE model is "pushed" from the *DSOL integrator* category to the *LIBODE vehicle* where the demand for integration is located. The data shows

[50] Eclipse Test & Performance Tools Platform www.eclipse.org/tptp/.

[51] According to the Eclipse TPTP Glossary, **base time** is the time spent executing a particular method, not including the time spent in other methods that this method calls.

VGF (vph)	15	22	30	40	50	60
LIBODE model	1094	1563	1953	2797	3782	4266
LIBROS model	344	469	563	688	828	953

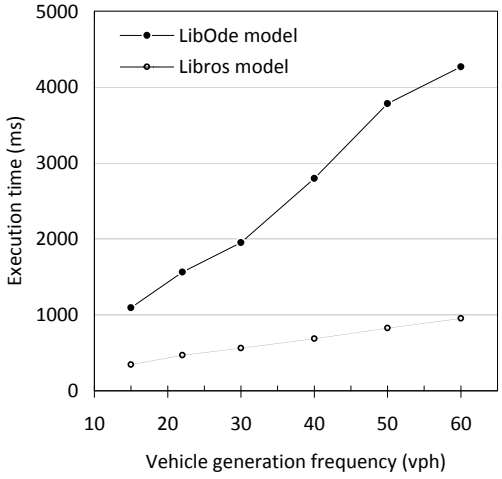


Figure 4.30: Execution time of the LIBODE and LIBROS tunnel models in milliseconds

the high computational demand of the integration as expected. The integration (or the vehicle model as the primary component in the LIBODE library) needs a large portion of the total execution time.

Compared with the *LIBODE vehicle* and *LIBODE track* categories, the *LIBROS vehicle/track* category is light-weight. But the *DSOL event-scheduling* in the LIBROS model is relatively heavy and the most demanding because all the DEVS model transitions and communications managed by the ESDEVS library eventually use the event-scheduler. Nevertheless, the data shows that the computational cost of this part is comparable to the cost of *DSOL event-scheduling* in LIBODE.

Note that the model communications in both libraries are handled on the cost of *DSOL event-scheduling*. The communication mechanism in LIBODE is based on publish-subscribe, in LIBROS it is message propagation (§ 4.2.3 and § 4.3.1). This means that, for the executed simulation experiments, the design of MP in LIBROS, despite extra (number of) message sending and state transitions, does not yield significantly more computation cost than the publish-subscribe communications in LIBODE. (In the case of 60 vph in the *DSOL event-scheduling* category, the total cost of transitions and communications in LIBROS is even lower than the cost of communications alone in LIBODE.) This result is consistent with the cost estimation of MP presented in § 4.2.3.

In general, the integrator and the use of it comprise the primary component for a

LIBODE model			
VGF (vph)	**15**	**30**	**60**
DSOL event-scheduling	140	237	505
DSOL integrator	175	256	444
LIBODE vehicle	539	1048	2383
LIBODE track	53	126	389
Others	188	285	544

LIBROS model			
VGF (vph)	**15**	**30**	**60**
DSOL event-scheduling	172	275	442
ESDEVS	16	43	88
LIBROS vehicle/track	6	15	31
Others	149	230	392

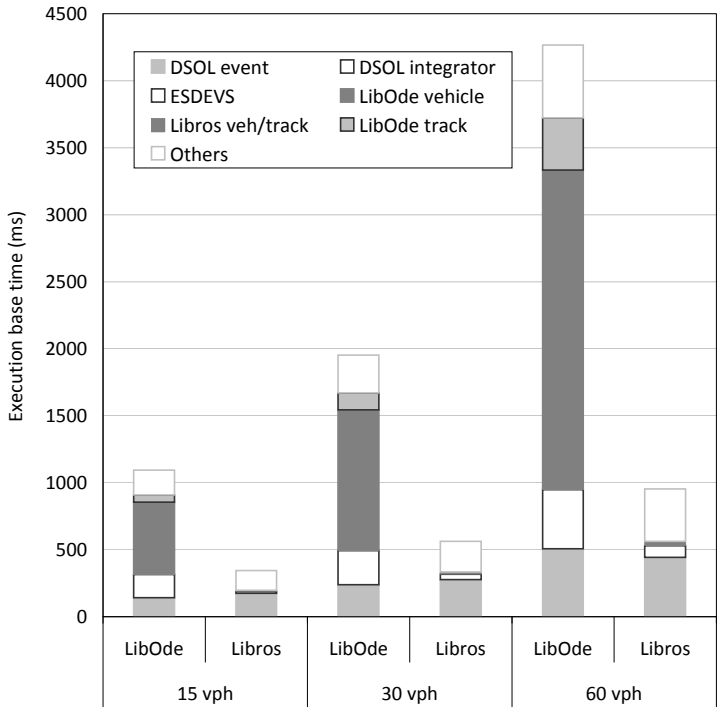


Figure 4.31: Execution base time per category of the LIBODE and LIBROS tunnel models

continuous model. The event handler (e.g., ESDEVS in combination with DSOL event-scheduler in case of LIBROS) and its use have comparable roles for a discrete-event model. Given the same integrator, the design choices of a continuous model, however, often have little influences on the efficiency of the numerical solution, since the integrator solves the differential equations with a defined time step. On the contrary, with a discrete-event model, the design choices (i.a., the choice of events and the state transitions on the events) often have influences on the time advances and hence affect model performance. This opens up possibilities for, but does not guarantee, improvement of model performance, and requires modelers to make careful choices of model design.

The results of the experiments show that, with comparable modeling details and accuracy, the LIBROS model yields higher performance than the LIBODE. This is an example that careful design of discrete-event models can yield high performance without loosing model accuracy.

4.5 Model/Simulation Presentation

As discussed in § 4.1.2, components for model/simulation presentation (MSP) take care of tasks such as model data output, visualization and statistics. In this section, we briefly present the MSP components in LIBROS and comment on some issues that are deemed important, particularly for modeling with DEVS. Figure 4.32 gives an overview of the MSP components in LIBROS. When a model is ran by a simulator, it becomes simulation. In this sense, the model and simulation are two aspects of the same thing. The former is a static system specification and the latter is the specified system in a dynamic running state. The two bold lines in the figure indicate the data flow during a simulation run. The data is being animated and recorded in a database.

Loose Coupling to Simulation Model The DSOL library (JACOBS 2005) that ESDEVS is based on has an animator as well. The animator *is a* simulator that supports *concurrent animation*[52]. When the simulator/animator is running a model, it can render the

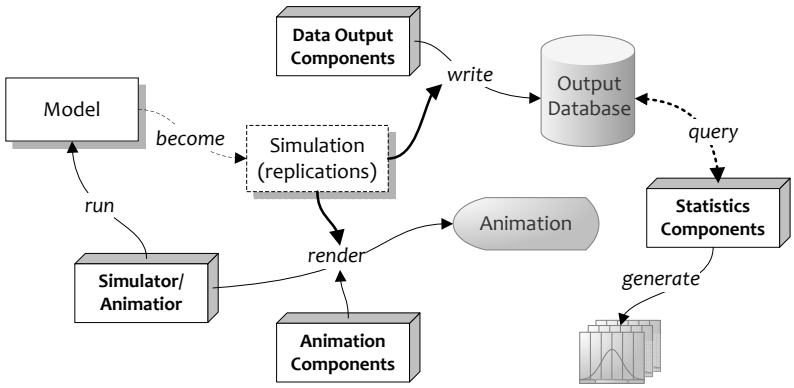


Figure 4.32: Model/simulation presentation components in LIBROS

[52] In **concurrent animation** the animation is being displayed at the same time that the simulation is running; in **post-processed animation** (or post-run animation), state changes in the simulation are saved to a file and used to drive the graphics after the simulation is over (LAW 2007). Post-run animation is not supported by the DSOL animator. A DSOL related work of post-run animation can be found in FUMAROLA (2011). Tools available for visualization of CD++ (a C++ based toolkit for DEVS and Cell-DEVS) models can be found in WAINER (2009).

animation of the simulation (at the same time) with Java 2D or 3D graphics. In order to do so, a modeler must provide the required rendering information about the 2D or 3D visual representation of the model and the location (and bound) of the visualization, by using `Renderable2D` or `Renderable3D` and `LocatableInterface` in DSOL.

Figure 4.33 shows an example in which a model component `Model1` can be animated by its 2D animation component `Model1_Visualization` using DSOL. The `Renderable2D` contains a `paint` function that defines how to visualize the model (state) at a certain instant. The function is invoked at each animation time step. The `Renderable2D` also contains a reference to the model component (i.e., the `source`) in order to access the latest model state. The model component on the other hand contains functions (i.e., `getBounds` and `getLocation`) that are related to the animation, i.e., these functions are also invoked at each animation time step. The approach results in tightly coupled model components and animation components.

To loosen the coupling, a *model image* is created for many (atomic) model in LIBROS to "portrait" the model's state. It is a small object that holds the values of a subset of the model's state variables used by the animation. The model image instead of

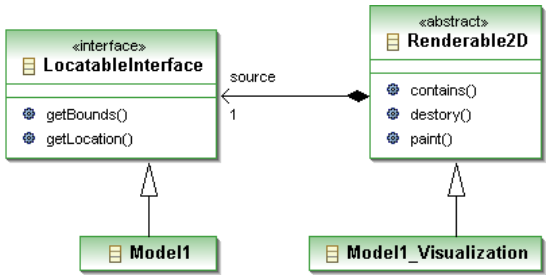


Figure 4.33: Model visualization with DSOL animator

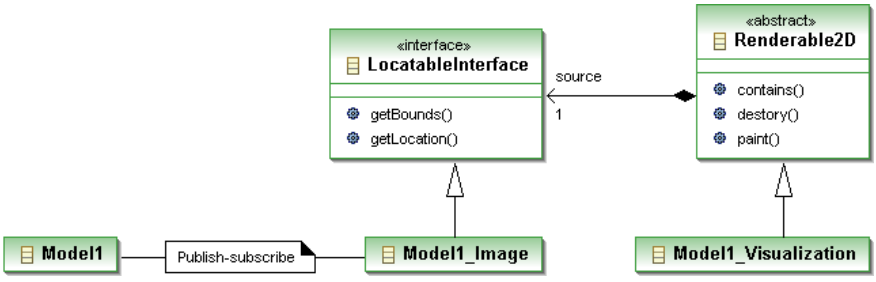


Figure 4.34: LIBROS model visualization with DSOL animator

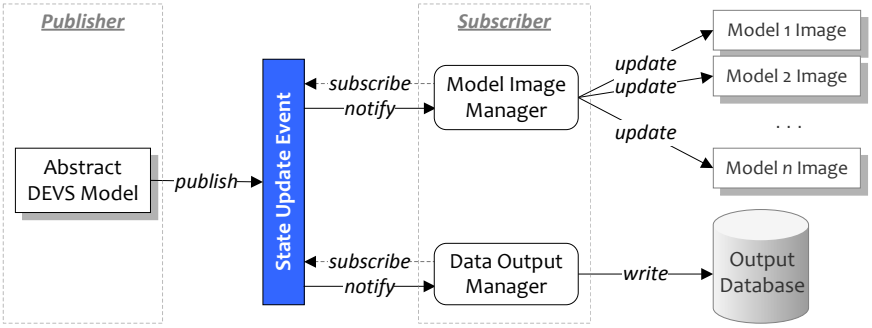


Figure 4.35: LIBROS model image update and data output with publish-subscribe

the model component itself is used with the corresponding animation component, as shown in Figure 4.34. With this small design adaptation, the animation code is moved away from the model components, and the animation components can only access the model images. During a simulation run, a model publishes its state changes, and its model image is subsequently updated with the changes through the publish-subscribe mechanism[53], as illustrated in Figure 4.35.

A *model image manager* takes the responsibility of updating all model images in a simulation run. It subscribes to the *state update event* of all models (i.e., the abstract DEVS model class) in LIBROS and is notified of the event each time a model state update takes place. The key benefit of using publish-subscribe in this context is that we can change either side the model components and animation components without much affect on the other. The disabling (or replacement) of animation can be neatly done by unsubscribing to the event (or the replacement of the subscriber).

For the same reasons, the publish-subscribe mechanism is also applied to the communications between the model components and the data output components, shown in Figure 4.35. A *data output manager* is notified of the model state updates and is responsible to prepare adequate database statements and subsequently write the data to the output database.

The statistics components[54] in LIBROS are responsible for querying data from the output database (after simulation), and display and run statistics on them. Based on user requirements, a number of statistics components are designed for KPIs such as halting time, trip time/speed, delay, regularity, punctuality, distance-time chart, etc., and the data can be aggregated per vehicle, per stop, and per service line.

[53] See § 4.4.1, fn. 49, p. 116.
[54] Called reporting and graphics in some literature; e.g., BANKS (1998) and LAW (2007).

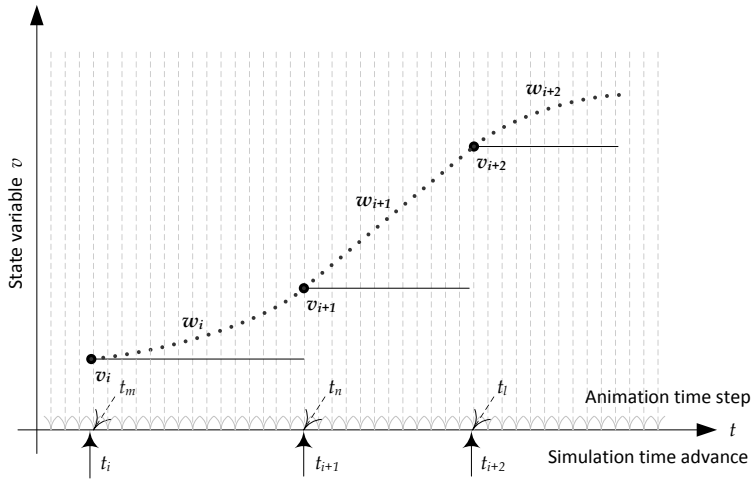


Figure 4.36: Smooth animation of discrete state variables

Animation of Discrete Event Models Concerning animating discrete event models, there is another issue we would like to discuss. The time advances of the simulation models are often much larger than the animation time step[55]. The direct visualization of the stepwise state changes of a discrete event model thus may cause undesirable flash or jump (LAW 2007) of the animated images. This is particularly true for GDEVS models since the coefficients of the polynomial segments are constant and the state variables often are not. Flashing or jumping images can result in erratic animation effects from an aesthetic point of view. But they can have a more serious negative effect. Because different discrete event model components do not have synchronized state transitions, this quality of animation may give misguided impression of the interrelation of the model states. When smooth animation is required, the presented state variables shall always represent the "current state" of the simulation (REKAPALLI 2008) along with the animation time advance. This means that their values need to be updated at each animation time step.

Let v be a state variable of a GDEVS model M and $w(t)$ be a piecewise polynomial representation of v, i.e., $v := w(t)$. Suppose that M has a state transition at time t_i, after which the polynomial becomes $w_i(t)$ and the value of v becomes v_i. With a non smooth animation, v remains constant at the animation time steps after t_i until the next state transition takes place, as illustrated in Figure 4.36. A smooth animation is able to update the value of v at the animation time steps by using the corresponding polynomial. Suppose that the animation time step right after t_i is at t_m, then the value

[55] The latter advances in small steps in a discrete time fashion.

of v at t_m is determined by $v_m = v_i + w_i(t_m - t_i)$; see § 4.3.3.2. Similarly, further values in this polynomial segment are determined by $v_{m+1} = v_i + w_i(t_{m+1} - t_i) = v_m + w_i(t_{m+1} - t_m)$, and so forth. The calculation is in a DTSS fashion (§ 2.4.1.2). When the next state transition arrives at t_{i+1}, the polynomial is updated to $w_{i+1}(t)$ (and v to v_{i+1}). The value at each animation time step is calculated with the new polynomial in the same manner.

Regardless whether smooth animation is required, the presentation of model states often need transformations related to spatial, geometrical, and/or temporal information, etc. These require careful design and are sometimes application specific. They can have high computational demand as well. We will not discuss related issues as they are not a focus of this thesis. Some examples of the animation in LIBROS can be found in § B.3.

Smooth animation is costly. Loose coupling between the model and animation is particularly desirable in such cases because one can easily disable the functions related to animation when it is so wished. In LIBROS, the state variables' values produced by the models for simulation are contained in the models, while the smoothened values for animation are produced by and contained in the corresponding model images. In combination with the publish-subscribe mechanism described earlier in this section, animation is activated only when the model image manager subscribes to the model state update event.

5 Model Generation

THE DOMAIN MODEL COMPONENTS discussed in § 4 are used for model generation. An AMG process in this research should automatically combine these components, instantiate and configure them so that they together constitute a simulation model. In § 3.4, we separated an AMG process into three AMG steps such that each step produces a major result. This chapter focuses on the first two steps: mode transformation and instantiation.

In principle, transformation rules for the AMG can be defined when the data for AMG input has semantic and pragmatic completeness, has definable measures for syntactic and mapping inconsistency (if any), and when the modelers have sufficient domain knowledge and deductive reasoning for the definition of transformation rules that can solve data issues related to semantic accuracy and presentation suitability (§ 3.3) with regard to model structure and parameterization.

Since graph theory and graph transformation in particular are applied for model transformation and instantiation in this research, the related theory is first presented in § 5.1. In the model transformation step explained in § 5.2, transformation rules are executed on the infrastructure data in order to construct a hierarchical graph that represents the compositional structure of the simulation model to be generated. The rules are defined on the meta-models of the original data structure (Construct 3), of the intermediate structures, and of the domain simulation model (Construct 1). In the model instantiation step explained in § 5.3, a simulation model is generated using the domain model components based on the hierarchical graph from the previous step. In § 5.4, we summarize the steps and sub-steps discussed in § 5.2 and § 5.3.

5.1 Graph Theory and Graph Transformation

In the AMG process, graphs are used to represent structures in data and model composition, to perform transformation step by step from the former to the latter, and to generate simulation models based on the latter.

5.1.1 Structure Representation with Graphs

Data can be structured, semi-structured or unstructured. In Example 3.1 given in § 3.2, the infrastructure data are unstructured. Simulation models, on the other hand, are often structured. The models discussed in this thesis have hierarchical structures.

This *hierarchical structure* refers to a *Compositional Containment Hierarchy* (CCH), which is a strictly nested component inclusion hierarchy. For example, component A is composed of components B and C in which C is composed of D and E, etc., as shown by the *Model Composite Tree* (MCT) in Figure 4.29.

Using a tree structure to represent CCH does not convey information about the interrelations (i.e., coupling relations in case of DEVS) among the model components, which comprises a graph (or network). For this reason, a hierarchical graph is needed to represent model interrelations with a hierarchical structure.

5.1.1.1 Graph and Graph Pattern

A common mathematical notion of a graph is a finite set of nodes (or vertices) among which some pairs of the vertices are connected by edges (or links). An enrichment of this notion is to type and/or attribute the vertices and/or edges. The following definitions are based on EHRIG et al. (2006a,b), GALLAGHER (2006) and FAN et al. (2010).

A **directed graph** (or **digraph**) is an ordered pair $G = (V, E)$ where V is a finite set of *vertices* and $E \subseteq V \times V$ is a set of *edges* in which $e = (v, v') \in E$ denotes an edge from vertex v to $v' \in V$. A **graph** hereinafter always refers to a digraph if not otherwise indicated.

A similar definition of graph is a 4-tuple $G = (V, E, s, t)$ where E is a set of edges without indications of the (source and target) vertex pairs. They are specified by the *source function* s and *target function* t such that $s, t : E \to V$.

A **path** p in a graph G is a sequence of vertices $(v_1, v_2, \cdots, v_n)$ such that $e_{i\in[1,n-1]} = (v_i, v_{i+1}) \in E \in G$.

A **typed graph** $TG = (G, T_V, T_E, \mathfrak{t}_V, \mathfrak{t}_E)$ has typed vertices and/or typed edges, where G is a graph, T_V and T_E are two finite sets of *vertex types* and *edge types*, and $\mathfrak{t}_V$ and $\mathfrak{t}_E$ are *vertex type function* and *edge type function* such that $\mathfrak{t}_V : V \to T_V$ and $\mathfrak{t}_E : E \to T_E$.

A **data graph** $DG = (TG, A_V, A_E, \mathfrak{a}_V, \mathfrak{a}_E)$ is a typed attributed graph, where TG is a typed graph, A_V and A_E are two finite sets of *vertex attributes* and *edge attributes*, and $\mathfrak{a}_V$ and $\mathfrak{a}_E$ are the *vertex attribute function* and *edge attribute function* such that $\mathfrak{a}_V : V \times A_V \to \mathbb{R}$ and $\mathfrak{a}_E : V \times A_E \to \mathbb{R}$. This means that a vertex $v \in V$ can have a number of vertex attributes $a_i \in A_V, i \in [1, n]$ each of which has an attribute value $c_i \in \mathbb{R}$, i.e., $\mathfrak{a}_V(v, a_i) = c_i$. Likewise, an edge $e \in E$ can have a number of edge attributes

$a_j \in A_E, j \in [1,m]$ each of which has an attribute value $c_j \in \mathbb{R}$, i.e., $\mathfrak{a}_E(v,a_j) = c_j$. As such, a vertex or edge can be assigned with a set of attributes and values that carry the content of the vertex or edge, e.g., label, rating, weight, and identifier.

Graphs can represent not only structure instances such as social networks or maps, but also types or patterns of structure instances. The token and type models illustrated in Figure 2.3, § 2.2.1, are examples. A type of graph structure instances can be expressed by a *graph pattern*. A graph pattern is often used to describe the matching criteria of occurrence(s) of homomorphic or isomorphic subgraph(s) in a host graph.

A (typed attributed) **graph pattern** can be defined as $P = (V_p, E_p, \mathfrak{p}_p, \mathfrak{b}_p)$ where V_p and E_p are two finite sets of vertices and edges, $\mathfrak{p}_p$ is the *predicate function* defined on V_p and/or E_p as a (logical) conjunction of atomic formulas[1] over the vertex and/or edge types T_V, T_E, the vertex and/or edge attributes A_V, A_E, and the corresponding attribute values, and $\mathfrak{b}_p$ is the *bound function* such that $\mathfrak{b}_p : E_p \rightarrow \mathbb{R} \cup \infty$. Note that an edge in a graph pattern is often a path in a host graph. Intuitively, the predicate function defines the search conditions on the vertices and/or edges in a host graph, and the bound function defines the bound of a search path in the host graph, e.g., the upper bound of a path length, and ∞ simply means that there is no bound.

5.1.1.2 Hypergraph

The definitions given in § 5.1.1.1 do not yet support the direct expression of CCH. In order to represent this hierarchy, we need a mathematical notion of composition. In literature, *hyperedges* (e.g., Busatto and Hoffmann 2001, Drewes et al. 2002, Palacz 2004, Bruni et al. 2010) are often used for this purpose.

A **hyperedge** is an edge in a *hypergraph*, a generalized graph whose edges are non empty sets of finite vertices. To the author's knowledge, it is first introduced by Berge (1973, 1989). Berge (1989) has the following definitions[2].

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a finite set of vertices. A **hypergraph** on V is a family $H = (E_1, E_2, \ldots, E_m)$ of subsets of V such that the edges $E_i \in H, i \in [1,m]$ satisfy $E_i \neq \emptyset \wedge \cup E_i = V$.
- A **simple hypergraph**[3] is a hypergraph such that no edge is contained by another, i.e., $E_i \subset E_j \Rightarrow i = j$.
- A **simple graph**[4] is a simple hypergraph such that each edge has cardinality 2, i.e., $|E_i| = 2$.

[1] Atomic formulas, as opposed to composite formulas, are the simplest well-formed formulas in mathematical logic. Variables and constants are (atomic) terms; if f is an operation of degree r and $t_1, \cdots, t_r$ are terms, then $f(t_1, \cdots, t_r)$ is a term; if p is a (predicate) relation of degree r and $t_1, \cdots, t_r$ are terms, then $p(t_1, \cdots, t_r)$ is an atomic formula, which has roughly the following meaning: the ordered r-tuple of objects denoted by $t_1, \cdots, t_r$ has the property denoted by the r-ary predicate p (Manin 2010, Ben-Ari 2012).

[2] A more detailed concept of hypergraph (and hyperedge) is presented, e.g., in Habel (1992) and Drewes et al. (1997) which include *tentacles*, *attachment vertices* and *external vertices*. We discuss them in § 5.1.2.4.

[3] Or *Sperner family*.

[4] The (ordinary) graphs that have edges connecting two vertices only, as those discussed in § 5.1.1.1.

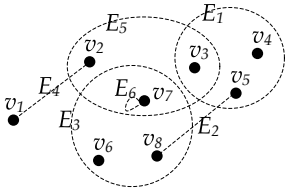


Figure 5.1: Hypergraph – an example with eight vertices and six edges (ibid.)

As proposed by BERGE (1989), a hyperedge E_i may be represented by a line connecting the two vertices if $|E_i| = 2$ (similar to the case in a simple graph), by a loop if $|E_i| = 1$, and by a closed circle enclosing the vertices if $|E_i| \geq 3$, as illustrated in Figure 5.1.

One can see that a hypergraph is not per definition a CCH or strictly nested. For example, a simple hypergraph does not have a CCH. A CCH shall not have hyperedges that can freely share vertices. Nevertheless, constrains can be imposed on the hyperedges in order to obtain CCHs by means of hypergraphs.

Hyperedges and Simple Edges In representing CCHs with hypergraphs, we need to pay attention to the hyperedges that have cardinality 2. In literature, a hyperedge connecting two vertices is like an ordinary graph edge. When we use these hyperedges to represent the compositional relations of two vertices, how to distinguish them with the coupling relations of two vertices?

Take the hyperedge E_2 in Figure 5.1 as an example. This edge connects v_5 and v_8. It can represent a compositional relation, where v_5 and v_8 are composed together to form a larger coupled component[5]; it can also represent a coupling relation, where v_5 and v_8 are simply coupled together. This ambiguity in relation definition is undesirable for model transformation.

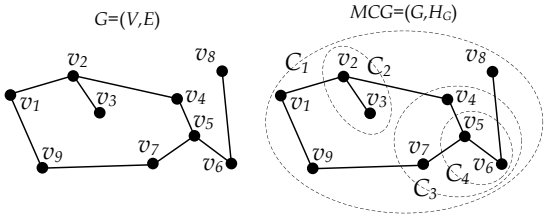
It would therefore make sense in our graph representation to distinguish hyperedges having cardinality 2 with simple edges, or at least type or label these edges, in order to represent compositional relations and coupling relations differently.

5.1.1.3 A Hierarchical Graph for Model Composition

Because a CCH is a strictly nested component inclusion hierarchy, we define the following constrain for the hyperedges: for each pair of hyperedges in a CCH, one edge can be a subset or superset of another edge but otherwise they must not intersect one another. Based on the discussions and definitions presented, we propose a definition of hierarchical graph that can be used to represent a CCH for model composition.

A **model composite graph** $MCG = (G, H_G)$ is an ordered pair of an ordinary digraph $G = (V, E)$ and a CCH H_G specified on G. The graph G can be typed and/or attributed

[5] This is of course impermissible in case of DEVS as both vertices are also contained by other hyperedges.



$H_G = (C_1, C_2, C_3, C_4)$
$C_1 = (v_1, v_8, v_9, C_2, C_3)$
$C_2 = (v_2, v_3)$
$C_3 = (v_4, v_7, C_4)$
$C_4 = (v_5, v_6)$

Figure 5.2: Model composite graph – an example with four compositions defined on a graph with nine vertices

as defined in § 5.1.1.1. The CCH $H_G = (C_1, C_2, \ldots, C_m)$ is a family of subsets of $V \in G$ such that the hyperedges $C_i, i \in [1, m]$ satisfy,

(1) $\cup C_i = V \ \wedge \ C_i \neq \varnothing$,

(2) $\exists C_i = V$, and

(3) $\forall C_i, C_j, j \in [1, m], i \neq j \implies C_i \cap C_j = \varnothing \ \vee \ C_i \subset C_j \ \vee \ C_i \supset C_j$.

We separate a graph G with its graph hierarchy H_G in the definition[6]. The model coupling relations are represented in the former by simple edges $e \in E$. The compositional relations are represented in the latter by hyperedges C_i which are sets of composite members only. The vertices $v \in V$ and the hyperedges C_i represent the (elementary and composed) components in model composition.

Figure 5.2 illustrates an example of MCG in which a hypergraph H_G with four compositions $C_1 \sim C_4$ are specified on a graph G with nine vertices $v_1 \sim v_9$. The solid lines denote simple edges and the dashed circles denote hyperedges.

Following the first condition in the definition (the same as in the hypergraph definition), each composition is a non empty set of vertices in graph G. The second condition states the existence of one component that contains all vertices in G. This is the root component that is to be instantiated as the top level model (§ 4.3.2.1) in AMG. The third condition in principle states that a CCH does not allow intersected compositions. A component can be a subset or superset of another component. Otherwise, they shall not share composite members. Note that the hyperedges can be as well typed and/or attributed following definitions in § 5.1.1.1.

5.1.2 Basic Concepts of Graph Transformation

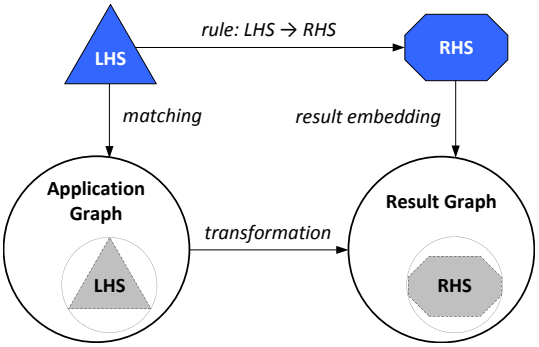
The main idea in graph transformation is rule-based graph pattern matching and rewriting, where the application of different transformation rules leads to different graph transformation steps (CORRADINI et al. 1997, EHRIG et al. 2006b).

[6] The definition is consistent with the hypergraph definition, e.g., in HABEL (1992) and DREWES et al. (1997); see § 5.1.2.4. The information about attachment vertices of the hyperedges is however contained (indirectly) in the ordinary graph G; see § 5.2.

5.1.2.1 Rules, Matches and Rule Applications

A graph transformation step involves the following basic concepts: graph transformation rules, matches and rule applications (CORRADINI et al. 1997, KAHL 2002). They are illustrated in Figure 5.3 (cf. Figure 3.6) and briefly discussed below (1997, 2002).

- A **rule** (or *production*) $p : L \rightsquigarrow R$ contains at least a **left-hand side** (LHS, or *pattern graph*) and a **right-hand side** (RHS, or *replacement graph*), and some indications of how instances of the RHS are to replace the matched instances of the LHS, e.g., which vertices and edges are to be preserved, deleted and/or created. The LHS and RHS are the source and target meta-models (or subsets of them) discussed in § 3.4.
- The application of a transformation rule to an **application graph** (or *host graph*) G requires a **match** (or *graph morphism*) $m : L \to G$ for a production p. A match occurs when the vertices and edges of L can be mapped to a subgraph in G such that the defined graph structure, bound and/or the types and/or the attributes (§ 5.1.1.1) are preserved. This subgraph is also called an *image*.
  Following CORRADINI et al. (1997), a **graph morphism** $f : G \to G'$ is a pair $f = \langle f_V : G_V \to G'_V, f_E : G_E \to G'_E \rangle$ of functions which preserve sources, targets, types and attributes, i.e., which satisfies $f_V \circ t^G = t^{G'} \circ f_E$, $f_V \circ s^G = s^{G'} \circ f_E$, $\mathfrak{t}^{G'}_V \circ f_V = \mathfrak{t}^G_V$, $\mathfrak{t}^{G'}_E \circ f_E = \mathfrak{t}^G_E$, $\mathfrak{a}^{G'}_V \circ f_V = \mathfrak{a}^G_V$, $\mathfrak{a}^{G'}_E \circ f_E = \mathfrak{a}^G_E$ (see § 5.1.1.1). A graph morphism is an *isomorphism* if both f_V and f_E are bijections. If there exists an isomorphism from graph G to graph H, then we write $G \cong H$. $[G]$ denotes the isomorphic class of G, i.e., $[G] = \{H \mid H \cong G\}$.
- The **rule application** to an application graph G, produces a **result graph** (or *derived graph*) H. The transformation relation (also know as *direct derivation*) is often called *co-production* $p^* : G \rightsquigarrow H$. Roughly speaking, H is constructed as $G \setminus (L \setminus R) \cup (R \setminus L)$. The rule application can be seen as an embedding into a context which is the part of the host graph G that is not part of the match. The **result**



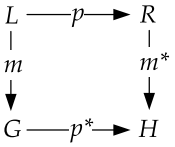


Figure 5.3: Graph transformation – rule-based modification of graphs (CORRADINI et al. 1997, KAHL 2002, EHRIG et al. 2006b)

embedding (or *co-match*) $m^* : R \to H$, maps R to its occurrence in the derived graph H.

Figure 5.3 (right) illustrates a schematic representation of a direct derivation from G to H, $G \overset{p,m}{\Longrightarrow} H$, resulting from an application of a production p at a match m (ibid.). A **graph grammar** $\mathscr{G}$ consists of a set of productions $\mathbf{P} = \{(p_i | i \in [1,n]\}$ and a *start graph* G_0. A sequence of direct derivations $G_0 \overset{p_1}{\Longrightarrow} G_1 \overset{p_2}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} G_n$ constitutes a *derivation* of the grammar, also denoted by $G_0 \Longrightarrow^* G_n$ (ibid.).

5.1.2.2 Comments on Graph Pattern Matching

Graph Pattern Matching (GPM) is to find one or all matches in an arbitrary host graph G for a given pattern P being the LHS of a rule (FAN et al. 2010). The problems in GPM can range from the subgraph isomorphism problem, near isomorphism, to the finding of inexact matches (GALLAGHER 2006). The variations of GPM generally occur along with the strict or non strict matches on graph properties, viz., restrictions on structures, types, and/or attributes (ibid.). As we will show in § 5.3.1, the GPM applied for the AMG in this research is an isomorphism problem rather than an inexact matching problem.

Complexity of Subgraph Isomorphism It is well know that the subgraph isomorphism problem is NP-complete (GAREY and JOHNSON 1979, EPPSTEIN 1999). In general, this problem can not be solved in sub-exponential time (ULLMANN 1976, EPPSTEIN 1999, GALLAGHER 2006). However, for some choices of G and P, better time complexities are possible (SKIENA 1997, EPPSTEIN 1999, FAN et al. 2010). For example, according to EPPSTEIN (1999), matching a fixed pattern P (with l vertices) can be solved in polynomial time[7], i.e., $O(n^l)$; matching a fixed pattern P in a planar G (i.e., planar subgraph isomorphism) can be solved in linear time, i.e., $O(n)$. Therefore, the choices of P and the type and pre-processing of G do have impact on the efficiency of the GPM algorithms.

Approaches for Time Complexity Improvement With regard to the choices of P, an ideal situation is to define patterns with possibly small sizes; the size refers to the vertex size l. An alternative approach (but same in principle) is to set bounds on the search paths when applicable. In general, type and/or attribute related constraints can significantly speed up the search (SKIENA 1997). Sometimes, due to the nature of the GPM problem, relative complex patterns need to be defined. We propose a graph pattern (matching) composite method that performs GPM step by step in an incremental bottom-up manner. Its principle is the same as using patterns with small sizes. The method is simple but effective for patterns that have recurrence of sub-patterns; more see § 5.2.3 and § 5.2.4.

[7] The induced subgraph isomorphism with a fixed pattern P can also be solved in polynomial time (CHEN et al. 2008).

The type (and content) of the host graph G in relation with that of the pattern P often gives good hints on how to pre-process G in order to render the search more efficient. The main goal of pre-processing[8] in this context is to reduce the search space and/or to fasten the search operation in the GPM algorithms. Common approaches in literature are *candidate selection* and *index construction* based on some summary information of the data graph G (WASHIO and MOTODA 2003). The summary information is computed on the given graph and is comprised of *graph invariants*, such as vertex degrees, counts of adjacent vertices, path lengths, which are quantities useful to characterize the graph (GALLAGHER 2006). The idea is that if two graphs (or subgraphs) are isomorphic, they must have the same invariants; as such simple comparison or classification may already be able to preclude non-matches before the detailed matching algorithms take place (WASHIO and MOTODA 2003, GALLAGHER 2006). Based on the invariant values, we may construct indexes into a graph (e.g., the vertices or some structure patterns) or partition the graph into non-overlapping sets of vertices in order to potentially speed up the GPM algorithms (SKIENA 1997, GALLAGHER 2006).

5.1.2.3 Comments on Graph Rewriting

An application of a production needs to rewrite a matched subgraph with the replacement graph defined in the production. Well discussed in literature, there are two problematic situations concerning graph rewriting (CORRADINI et al. 1997, KAHL 2002).

Conflict in a match Because matches in general do not have to be isomorphic, there can be conflicts between deletion and preservation of elements in a match.

Dangling edges A vertex that shall be deleted is connected to an edge that is not a part of the match.

Two common approaches to graph rewriting in literature are the so called *double-pushout* (DPO) and *single-pushout* (SPO), whose basic difference lies in the way they handle the above two situations (CORRADINI et al. 1997, EHRIG et al. 1997). The latter being a more recent approach allows to apply productions without any application conditions, i.e., deletion has priority over preservation[9]; while the former has a build-in application condition, i.e., rewriting is not allowed in the above two situations (CORRADINI et al. 1997). EHRIG et al. (1997) shows how to extend the SPO approach to handle user-defined application conditions; thus the DPO approach could be considered as a special case of SPO (ibid.). We therefore choose to use the SPO approach for the model transformation in the AMG for three reasons:

(1) The GPM algorithms for the generation of MCG for LIBROS models only return isomorphic matches.

(2) Deletion is not performed in the productions for the generation of MCG.

[8] The pre-processing of G can also deal with tasks such as data cleaning related to the data quality of G. This is discussed in § 5.3.1, but not relevant to the discussion of pre-processing of G in this section.

[9] This is adequate in some application areas and problematic in other ones (EHRIG et al. 1997).

(3) In case of changes in the above conditions, the SPO approache allows for extensions of user-defined application conditions.

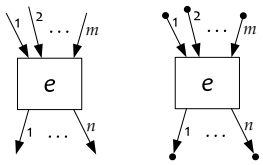
5.1.2.4 Hyperedge Replacement

Hyperedge replacement is the replacement of hyperedges of a hypergraph by hypergraphs (HABEL 1992). Following HABEL (1992) and DREWES et al. (1997), a directed hyperedge can be seen as a black-box or a placeholder with an ordered set of *incoming tentacles* and an ordered set of *outgoing tentacles*, as shown in Figure 5.4-(A). HABEL (1992) gives the following definitions.

- A hypergraph H with a finite set of vertices V and a finite set of hyperedges E has source and target functions $s : E \to V^*$ and $t : E \to V^*$ [10] that assign a sequence of sources $s(e)$ and a sequence of targets $t(e)$ to each $e \in E$.
- There is a (predefined) set of vertices occurring in the sequence $ext_H \in V^*$ which is called the set of **external vertices** of H. They (also called the *begin and end vertices*) correspond to the sequence of sources and targets of a hyperedge ($\notin E$) when H may replace this hyperedge (conditions see below). The set of all other vertices is said to be the set of **internal vertices** of H.
- The set of vertices occurring in the sequence $att(e) = s(e) \cdot t(e)$ is called the set of **attachment vertices** of an hyperedge e.
- A hyperedge $e \in E$ is called an (m,n)-edge for some $m,n \in \mathbb{N}$ if $|s(e)| = m$ and $|t(e)| = n$. The pair (m,n) is the *type* of e, denoted by $type(e)$.

Figure 5.4-(B) illustrates a hyperedge where $type(e) = (m,n)$ whose attachment vertices (i.e., source and target vertices) are denoted by dots (•).

Given hypergraphs H and R, a hyperedge $e \in E \in H$ may be replaced by R when e and R "fit together"; this means that when e and R are of the same type — if e is an (m,n)-edge then R is an (m,n)-hypergraph — and whenever the i-th and the j-th external vertices of R are the same then the i-th and the j-th attachment vertices of e are the same (i.e., distinct tentacles of a hyperedge may be attached to the same vertex, ibid.). The hyperedge replacement can be done by removing the hyperedge e, adding



(A) (B)

(A) A hyperedge with m incoming tentacles and n outgoing tentacles

(B) A hyperedge with m sources and n targets

Figure 5.4: Hyperedge of type (m,n) (ibid.)

[10] For a set A, A^* denotes the set of all strings over A including the empty string.

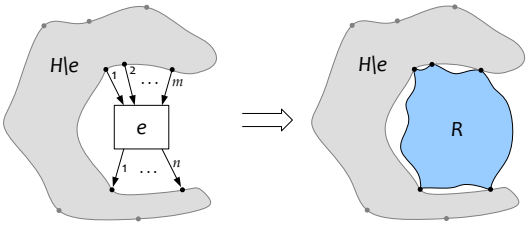


Figure 5.5: Hyperedge Replacement $H \Longrightarrow H[e/R]$ (HABEL 1992)

the hypergraph R except its external vertices, and handing over each tentacle of each hyperedge (in the replacing hypergraph R) which is attached to a begin or end vertex to the corresponding source or target vertex of the replaced hyperedge (1992). Figure 5.5 illustrates the replacement construction.

A formal definition of hyperedge replacement can be found in HABEL (ibid.). ANDERSSON (2006) provides a simpler definition as following. Let H, R be hypergraphs, $e \in E \in H$, $type(e) = type(R)$. The replacement of e in H by R yields the hypergraph $H[e/R]$ which is obtained with three steps:

(1) build $H \backslash e$ by removing e from H;

(2) take the disjoint union of $H \backslash e$ and R, i.e., $H \backslash e \uplus R$;

(3) $\forall i \in [1, type(e)]$, identify the i-th external vertex of R with the i-th attachment vertex of e.

This means that when adding R to $H \backslash e$, the sequence of external vertices of R is fused with the sequence of attachment vertices of e in the right order (DREWES et al. 1997).

5.2 Model Transformation

The model composition for the AMG of LIBROS models is derived from the AutoCAD data of the rail infrastructure layout, e.g., an intersection or some area, under modeling interest. The data used for the studies are provided by HTM (§ 4.1.1). It was originally produced as light-rail infrastructure blueprints of the *Haaglanden* region. Figure 5.6 shows a plot of the infrastructure of the region. (The plot in Figure 3.2 of Example 3.1 is a small subset of it.) As stated in Example 3.1, the data is unstructured and contains a list of geometric primitives each of which describes a CAD entity.

With a CAD file as a start graph G_0 (§ 5.1.2), taking into consideration the data quality issues discussed in § 3.3 (particularly in § 3.3.3), a set of productions (or transformation rules) **P** is defined to derive a model composite graph MCG (§ 5.1.1.3) for the AMG of LIBROS models in the model transformation step. These transformation rules are presented in this section. (The final production from the MCG to a LIBROS simulation model is presented in § 5.3.)

5.2.1 On Start Graph

The CAD infrastructure data contains a list of CAD entities each of which has a geometric shape with geometrical descriptions and possibly other descriptions such as color and label (see, e.g., Figure 3.2). Although the CAD data plot visually appears to be a network, the dataset itself does not contain descriptions about the relations of the entities. As such, it is hardly a graph as a mathematical structure. When considered as a graph, it has vertices which are the CAD entities but has no edges. The shape type of a CAD entity is the vertex type, and the descriptions (including geometry and others) are vertex attributes.

A model composite graph $MCG = (G, H_G)$, on the other hand, is structured. It needs to provide sufficient information about the composite of the simulation model to be generated. First of all, G shall be a digraph where the edges represent (directed) model coupling relations. In case of LIBROS models, each edge indicates the direction of the traffic that consists of two coupling relations (§ 4.3.4.1). Second, the hyperedges in the MCG shall describe the structure of each component. They are the results of GPM where different types of infrastructure layouts are defined as graph patterns.

5.2.1.1 Infrastructure Descriptions in Start Graph

The entities in the infrastructure CAD data are mainly lines and arcs that represent rail tracks. There are also labeled circles that indicate the start locations (and lengths) of the stops. From their geometric properties such as world coordinates and shapes, we can determine the connectedness of the entities.

The CAD entities were drew manually as rail track blueprints. The major data quality

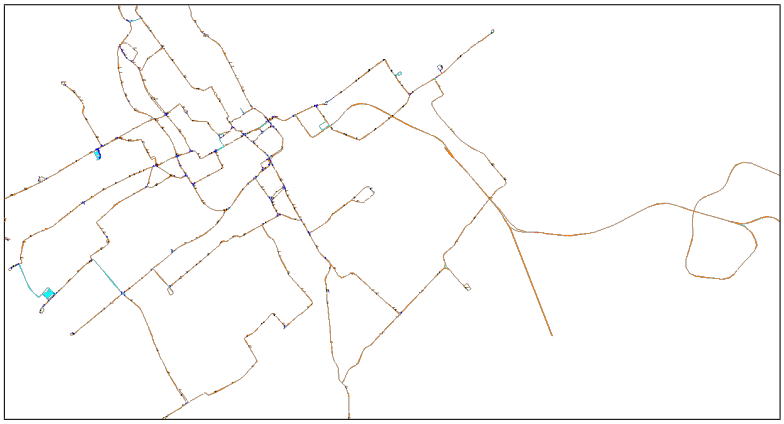


Figure 5.6: Light-rail infrastructure CAD data of the *Haaglanden* region

issues in this regard concern *semantic accuracy* (i.e., criterion #3, § 3.3.3).

(1) In a graphical CAD environment, two entities may appear to be connected visually when zoomed out with a relative high scale factor but are indeed unconnected[11] which can be inspected only when zoomed in with a sufficiently low scale factor. As such, the "connected" parts would not have the same world coordinates.

(2) For similar reasons, there exist some small "invisible" entities, e.g., very short lines and arcs, which shall not be considered as a part in graph transformation. For geometry-based inference of entity interrelations, these small entities may lead to misinterpretation[12].

(3) Although the entities representing rail tracks have start and end points, they do not correspond to the directions of the traffic.

In order to solve these issues, the following measures are applied. For issue (1), a configurable parameter for *snap tolerance* is used[13]. This specified distance is however not for correcting or modifying the coordinates of "connected" parts (because the aperture size is so small that it is not significant for distance calculation) but only for evaluating the connectedness of the CAD properties.

For issue (2), we "traverse" the infrastructure entities for connectedness with the defined snap tolerance. As such, the standalone entities (including the small entities) are ignored. Among the connected ones, a small entity[14] that only has one connected end point is ignored.

As for issue (3), the direction of traffic is not an intrinsic track property that can be inferred from the rail track geometry alone. However, if the origin of a traffic flow can be indicated in some way, and given that a rail track has a unique direction of traffic, it is possible to infer the direction of traffic of all the successively connected tracks. Urban light-rail services largely operate on rails with dedicated directions (PACHL 2002, VUCHIC 2005); this is the case in the *Haaglanden* region. The origins of traffic flows would typically either be at the boundary of the modeled area or at some defined locations such as terminals in the area. In LIBROS models, they are represented by the `source` components (§ 4.3.5.4). Since there is a low number of sources[15], it is affordable to manually specify where the sources are located. We did so by adding this information into the original infrastructure CAD data using labeled circles[16]. Figure 3.2 shows examples of these labeled source circles.

[11] This is can be caused, e.g., by a misuse of auto-snap as a drawing aid.

[12] In case of the LIBROS models, problems can occur when, e.g., a small entity is connected to two other track entities, by which a normal one-to-one track entity connection will be mistakenly interpreted as a rail switch.

[13] Many CAD tools and drawing tools have such a parameter; it is sometimes called, e.g., snap sensitivity or aperture size.

[14] An entity is considered to be small when its length is within a predefined value. This value is by default the same as the snap tolerance.

[15] For example, there are three sources in Figure 3.2 of Example 3.1 and fifteen in the whole *Haaglanden* region (Figure 5.6).

[16] They are labeled *Block* entities in AutoCAD named as "source".

5.2.1.2 Information Types and Dependencies

With the basic measures in § 5.2.1, we start out to design the intermediate model transformation steps. The major data quality issues that shall be solved by the transformation steps concern *presentation suitability* (i.e., criterion #8, cf., § 3.3.3). To solve the information gap, the types of information that are available in the CAD data are enumer-

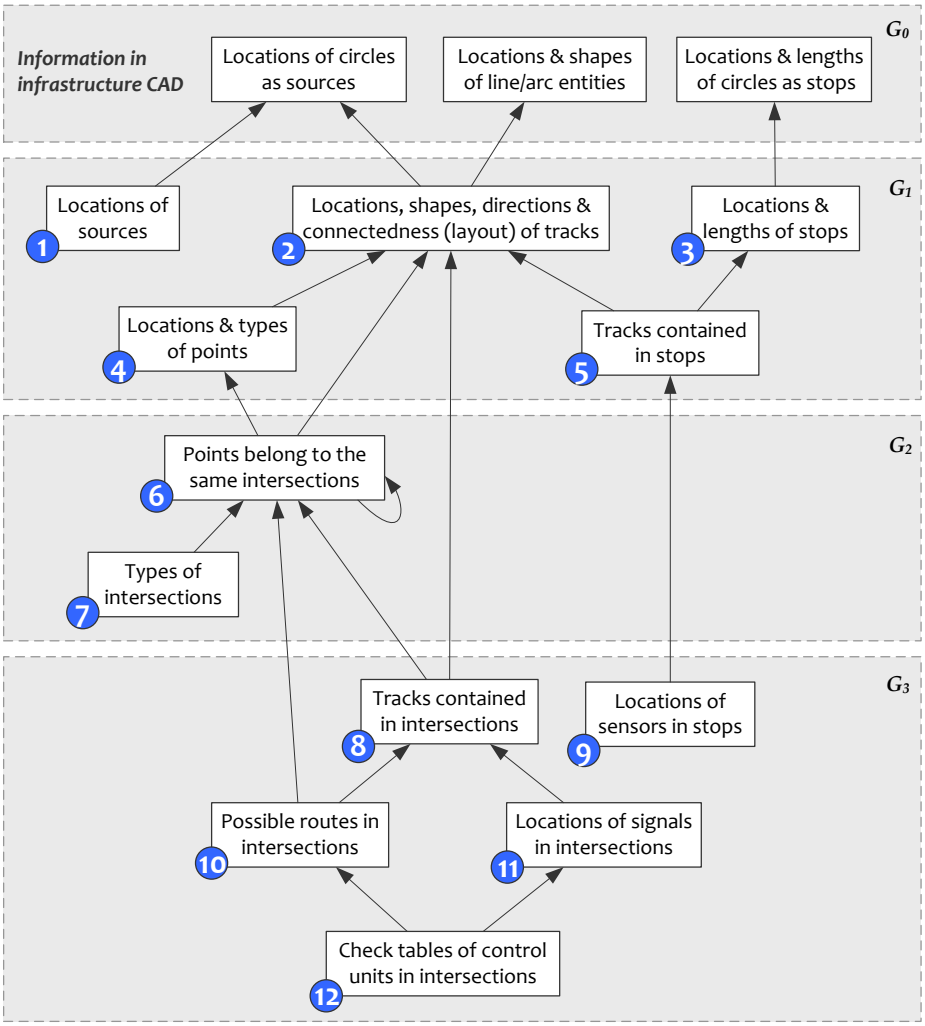


Figure 5.7: Composite information types and dependencies for the AMG of LIBROS models

ated against the information that is required for the MCG of LIBROS models. We then try to arrange the information types (Info. Types) by their dependencies and to fill in the gaps by adding information that can be inferred from those which are known or can be known. Expert opinions and literature are consulted in order to complete and verify the information dependencies and inference logic. A number of reviews and modifications took place (during the design and also the development process). In the final version, twelve information types are needed for the AMG of LIBROS models. Their dependencies are illustrated in Figure 5.7. We arrange the information into groups, each of which represents an intermediate transformation step. The information in one group can be either inferred within one single graph traversal, or the types are so interrelated that we choose not to divide them into different steps. The rest of this chapter explains how the information is obtained through stepwise transformation.

5.2.2 Transformation Step 1

The start graph G_0 is composed of a list of CAD entities. The first transform step shall construct a digraph G_1 whose vertices are of two types: sources or tracks (Info. Types 1 and 2). The geographical compositions of the tracks are indicants of locations and types of points (Info. Type 4). Additionally, since a stop model contains a sequence of connected tracks with a suitable total length (see e.g., § 4.4.1) and the required information is already available at this step, hyperedges are defined on G_1 to represent the stops and their constituent tracks (Info. Types 3 and 5).

We pre-process[17] G_0 such that its entities are partitioned into three non overlapping *sets*, viz., sources, tracks and stops[18]. The construction of G_1 basically relies on geometrical inference. The rail infrastructure has a number of intrinsic characteristics that allow for a limited number of direct CAD entity-to-entity compositions in terms of vertex types and degrees. Figure 5.8 illustrates the schema of these compositions. Note that the white headed arrows (as in the figures hereafter) represent track entities which are vertices but not edges, and their directions indicate permissible directions of the traffic which are not contained in the CAD data as such but need to be inferred during transformation.

The transformation in this step is completed with one "graph traversal" during which the graph edges must to be created based on the entities' geometrical connectedness. We apply *Depth-First Search* (DFS) for the traversal since DFS can be used to classify the edges (CORMEN et al. 2001) which suits our situation of exploring G_0. In the traversal, the search root of a new (depth-first) tree in the depth-first forest (ibid.) is always a source vertex.

5.2.2.1 Search For Connectedness

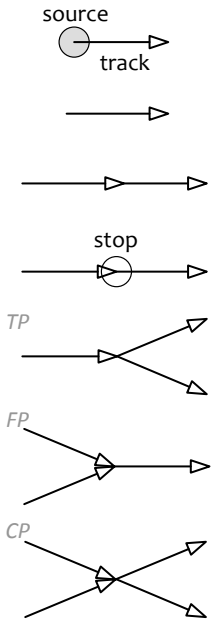
The search of CAD entities' geometrical connectedness to a reference point takes place (along the traversal) with a given snap tolerance; both the start and end points of a

[17] The benefit of pre-processing and common approaches are discussed in § 5.1.2.2.
[18] Their properties in the CAD data can be found in § C.1.

track entity are checked for connectedness[19]. In the search result, if there is a very short connected track which has nothing else connected to it, it is simply ignored[20].

An ordered pair of connected entities is an edge in digraph G_1. A discovered/visited edge, which would be a (source, track) pair or a (track, track) pair, is recorded in a map-like data structure, let us call it *track map* **T**, in which the leading vertex is an index (or key) and the following (track) vertex is an indexed value[21].

Representing Graphs In literature, there are two standard ways to represent graphs: using *adjacency/incidence-lists* or *adjacency/incidence-matrices* (ibid.). Since infrastructure data has potentially a large number of entities and the corresponding graph is sparse (i.e., $|E| \ll |V|^2$), matrices are not good choices because of its $\Theta(|V|^2)$ memory (ibid.). But adjacency/incidence-lists do not have fast edge search (ibid.). The track map we use is adjacency-list alike while other data structures are used to support fast edge search. These data structures are discussed in the following paragraphs.



(A) A source connected with a track

(B) A track without a descendant

(C) Two connected tracks

(D) Two connected tracks with a stop in between
*a composition enhanced on (C)

(E) Three tracks with one shared connecting point
*one track leads to two other tracks

(F) Three tracks with one shared connecting point
*two tracks both lead to the third track

(G) Four tracks with one shared connecting point
*tracks are paired, one only leads to one other track

Figure 5.8: CAD entity-to-entity compositions

[19] See § 5.2.1, data quality issues (1) and (3).
[20] See § 5.2.1, data quality issue (2).
[21] An index can be associated with more than one value.

5.2.2.2 Search From A Source

Starting from a source vertex, the connected track vertices are searched in the track set. A valid source shall have exact one track connected to it[22], shown in Figure 5.8 (A). This track entity (and any other track entities) shall have the start and end points correspond to the permissible direction of the traffic; if not, the these two points must be swapped[23]. We call this operation *regulating track direction*. A track with a regulated direction is called a *regulated track*. The DFS continues with the found regulated track.

5.2.2.3 Search From A Regulated Track

Starting from a regulated track vertex (with its end point as the reference point), the connected track vertices are again searched in the track set. We can encounter four (valid) situations regarding the search result: the regulated track has *non*, *one*, *two*, or *three* connected tracks.

Non The first situation, non connected track as shown in Figure 5.8 (B), is where a sink model shall be later placed. The DFS cannot go deeper at the current branch and must proceed with the closest non-fully-explored track vertex (in the current depth-first tree) if any, or with the next source vertex if any, or otherwise the traverse terminates, so does this transformation step.

One In the second situation, Figure 5.8 (C/D), the found connected track is first regulated and then the (track, track) pair is added into **T**. This is the only situation where a stop entity can be present, so a search is performed in the stop set. If a connected stop is found, a hyperedge shall be created to represent the stop. In either case, the DFS continues with the found track.

A stop entity indicates the front, i.e., start location, of a stop, as shown in Figure 5.9 (c.f. Figure 3.2). Each stop has a defined length[24] l_s. The track vertices that fit into this

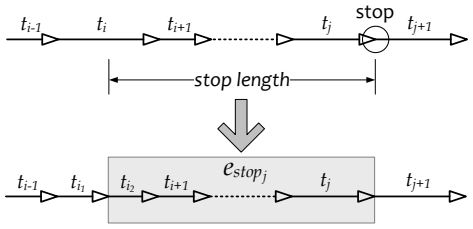


Figure 5.9: Creating a stop hyperedge

[22] A source without any track connected to it is simply ignored.
[23] In case the track is an arc, the start and end angles must also be swapped besides their coordinates.
[24] It is indicated by a label of "halting place number" which is often 1 or 2, see e.g., Figure 3.2. Each HTM halting place is 37 meters in length.

length shall be included in a "stop hyperedge". Suppose that the traversal is at (t_j, t_{j+1}) ($j \in \mathbb{N}$ which is numbered in the order of the traversal) and a stop entity is found. The track vertices that belong to the hyperedge are the just visited vertices. As such, we need to trace back into the visited vertices[25], i.e., $t_j, t_{j-1}, \cdots$, until, say t_i ($i \leq j$), so that the total length of the vertices can "cover" the stop length. Since the lengths unlikely fit exactly, we would need to "split" t_i into two, say t_{i_1} and t_{i_2}, the second of which shall fit the stop length (i.e., $l_{t_{i_1}} = \sum_{k=i}^{j} l_{t_k} - l_s$ and $l_{t_{i_2}} = l_{t_i} - l_{t_{i_1}}$). Their geometric properties such as the coordinates of start and end points (and start and end angles in case of arcs) have to be configured accordingly. Here, we have a simple though first true sense of graph rewriting, in which vertex t_i is replaced by two vertices t_{i_1}, t_{i_2} and one edge (t_{i_1}, t_{i_2}). Assume that t_{i_1} and t_{i_2} are already created as mentioned above, then we can rewrite graph G_1 as following:

1 REMOVE $(t_{i-1}, t_i), (t_i, t_{i+1})$ in $\mathbf{T}$
2 WRITE $(t_{i-1}, t_{i_1}), (t_{i_1}, t_{i_2}), (t_{i_2}, t_{i+1})$ in $\mathbf{T}$

A hyperedge e_{stop_j} composed of vertices $(t_{i_2}, t_{i+1}, \cdots, t_j)$, or only vertex t_{i_2} in case of $i = j$, is created to represent the stop. According to HABEL (1992), it is a $(1,1)$-edge with t_{i_1} and t_{j+1} as the (source and target) attachment vertices[26]. All created hyperedges are recorded in a *hyperedge map* $\mathbf{E}$, which is discussed in § 5.2.2.4.

Two A regulated track t_j with *two* connected tracks t_{j+1}, t_{j+2} can form two distinct track compositions as shown in Figure 5.8 (E/F), cf. Figures 4.17 (D/E) and 4.24 (C/D), in which t_j always points towards the "center" of the composition. In case of a facing point (FP) composition, t_j connects "smoothly" to both t_{j+1} and t_{j+2}. In case of a trailing point (TP) composition, only one of the two connections is smooth; see, e.g., Figure 4.20 (A). Note that the tracks themselves are not constituents of the point models which will be later generated. They only indicate the point location by their "center".

Suppose we are at the FP case, i.e., Figure 5.8 (E), then t_{j+1} and t_{j+2} are regulated such that both are pointing away from t_j. The (t_j, t_{j+1}) and (t_j, t_{j+2}) pairs are added into $\mathbf{T}$, and the FP is recorded by adding t_j (we call it *facing track*) into an *FP set* $\mathbf{P_F}$. The DFS now has two possible branches, one of which is taken to continue the traversal, and the other goes into the traversal stack.

Suppose we are in the TP case, i.e., Figure 5.8 (F), and t_j connects smoothly to t_{j+1}. We regulate t_{j+1} and add the (t_j, t_{j+1}) pair into $\mathbf{T}$. In addition, the (t_{j+1}, t_j) pair is added into a *TP map* $\mathbf{P_T}$. The TP map is used to record the found TPs, in which the following (track) vertex is an index and the leading one is an indexed value. This means that the index-value pair is *reversed* as that in $\mathbf{T}$, and when the DFS is terminated, each index in $\mathbf{P_T}$ has exactly two values. The DFS continues with t_{j+1} (we call it *trailing track*). Note that t_{j+2} is left untouched, because it either was visited or will be visited by the traversal.

[25] A solution to have a fast trace back is to maintain a small vertices buffer.
[26] We hereinafter call the sources or source vertices of hyperedges (HABEL 1992) as source attachment vertices to differentiate them with the source vertices in ordinary graphs, such as G_1.

Three With *three* connected tracks $t_{j+1}, t_{j+2}, t_{j+3}$, the only valid track composition is a *crossing point* (CP), Figure 5.8 (G), in which the regulated track t_j has only one smooth connection, say to t_{j+1}; see, e.g., Figure 4.20 (A). The same as in the TP case, we regulate t_{j+1} and add the (t_j, t_{j+1}) pair into $\mathbf{T}$. Of the two remaining tracks t_{j+2}, t_{j+3}, which one is leading to the other only can be known after these two vertices were already visited. If they were, there shall be a (t_{j+2}, t_{j+3}) or (t_{j+3}, t_{j+2}) entry in $\mathbf{T}$. Suppose that an entry can be found, say (t_{j+2}, t_{j+3}), then we can record this CP into a *CP map* $\mathbf{P_C}$. The CP is recorded such that each of the two leading tracks is indexed by the other leading track. This means that each CP shall have two entries in $\mathbf{P_C}$. The two entries in this case are (t_j, t_{j+2}) and (t_{j+2}, t_j). The DFS continues with t_{j+1}.

Remark The DFS proceeds each time when there is a connected track and terminates when the conditions are met as described in the first situation (i.e., when there is non connected track). After the traversal, the track map $\mathbf{T}$ holds the information about the (source) track interrelations (Info. Types 1 and 2); together with the FP set $\mathbf{P_F}$, TP map $\mathbf{P_T}$ and CP map $\mathbf{P_C}$, they hold the information about the points (Info. Type 4). Figure 5.10

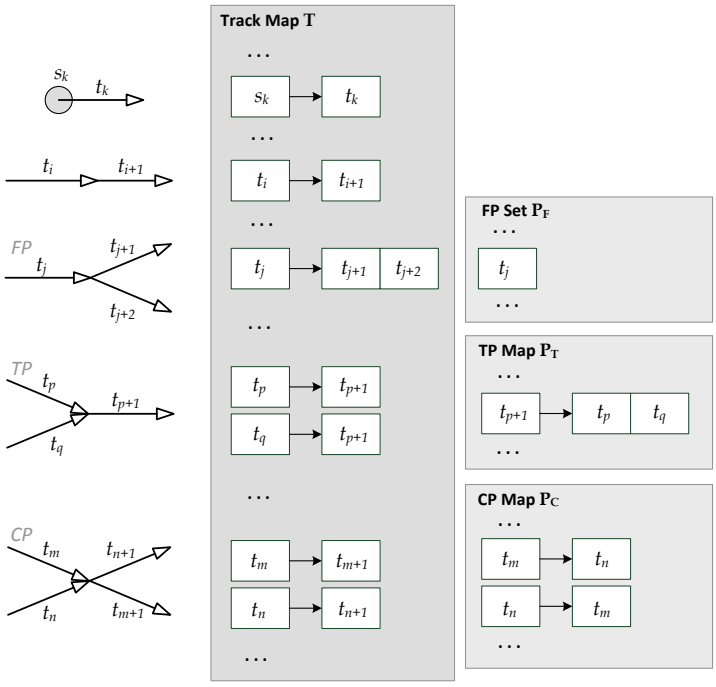
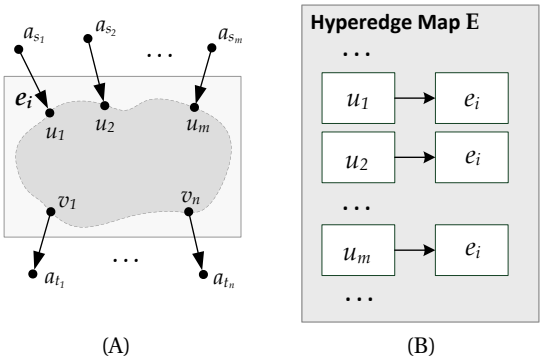


Figure 5.10: Representation of track composition information in set and maps



(A) (B)

$A_s = (a_{s_1}, a_{s_2}, \cdots, a_{s_m})$
$A_t = (a_{t_1}, a_{t_2}, \cdots, a_{t_n})$

$B_{en} = (u_1, u_2, \cdots, u_m)$
$B_{ex} = (v_1, v_2, \cdots, v_n)$

Figure 5.11: An (m,n)-edge representing a composite with m entry vertices and n exit vertices. It has m entries in the hyperedge map

illustrates the representation of the information in set and maps, with which which G_1 is described. The hyperedge map **E** (§ 5.2.2.4) shall hold information about all (potential) components; at this stage it only holds information about the stops (Info. Types 3 and 5). As the outcome of transformation step 1, G_1 can be expressed such $G_1 = (\mathbf{T}, \mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E})$.

5.2.2.4 Hyperedge Representation of Composition

Let e be an (m,n)-edge with a set of source attachment vertices $A_s = (a_{s_1}, a_{s_2}, \cdots, a_{s_m})$ and a set of target attachment vertices $A_t = (a_{t_1}, a_{t_2}, \cdots, a_{t_n})$. The edge e can be replaced with a given (m,n)-hypergraph, say H_e (§ 5.1.2.4). When we want to generate this H_e, we need to at least know the composition of H_e, without saying that the external vertices of H_e shall match the attachment vertices of e. This composition is described as a subgraph in G_1. We need to yet still define the "boundary" of this subgraph.

Can we use the attachment vertices of e as the boundary of e? Certainly we can. But since the attachment vertices potentially have edges with vertices that shall not be a part of H_e, we would need extra operations to check this through. Additionally, these attachment vertices can take part in other hyperedges. It is conceptually not quite explicable to use members of some hyperedges as boundaries of some other hyperedges. Therefore, we use an equally easy but more straightforward way to define the boundary, that is to use the "outermost vertices in H_e". They are the vertices that shall[27] belong to H_e and have edges with vertices that shall be external vertices of H_e. We call these vertices *boundary vertices* B of hyperedge e, where $B \in V \in e$. The boundary vertices that are connected by source attachment vertices are *entry vertices* B_{en}; those that are connected to target attachment vertices are *exit vertices* B_{ex}, i.e., $B_{en} \cup B_{ex} = B \in V$, $B_{en} \cap B_{ex} = \varnothing$.

Figure 5.11 (A) illustrates an (m,n)-edge with corresponding attachment and boundary vertices. Although the figure only shows one-to-one attachment-boundary vertex

[27] We use "shall" because at the stage when we define the boundary, the hyperedge H_e does not yet exist.

connections for simplicity, this is not a vital condition. An attachment vertex can connect with more than one boundary vertices and vice versa. The two sets B_{en} and B_{ex} specifying the boundary of a hyperedge e are needed for generating the replacement graph H_e. We therefore extend the hyperedge definition in the MCG (§ 5.1.1.3) with these two sets, while the incoming and outgoing tentacles connecting A_s with B_{en} and B_{ex} with A_t respectively are already specified in the track map **T**.

Hyperedge Map As mentioned in § 5.2.2.3, the hyperedges created during the transformation steps are recorded in a hyperedge map **E**. The edges are indexed by their entry vertices such that for *each* entry vertex $u_i \in B_{en}$ in a hyperedge e, there exists one map-entry (u_i, e) in **E**. This means that the entry vertices are not used as joint but independent indexes, and each (m,n)-edge has m entries in **E**, as illustrated in Figure 5.11 (B). The stop hyperedge e_{stop_j} discussed in § 5.2.2.3, for example, has only one entry (t_{i_2}, e_{stop_j}) in **E**.

Note that a hyperedge is defined by its entry and exit vertex sets together with the composite information about all the internal vertices which is already contained in G_1, i.e., the set and maps discussed in § 5.2.2.3. The hyperedge map **E** serves to fasten the search of hyperedges during pattern composite (§ 5.2.3.2) and model generation (§ 5.3). When the search is in the order of the traversal, or in the direction of the traffic in our case, it is sufficient to index hyperedges with entry vertices. If modelers need to search in a reversed order, hyperedges can be indexed by their exit vertices.

5.2.3 On Model Composite Graph

As discussed in § 5.1.1.3, the MCG is composed of an ordinary digraph G and a CCH hypergraph H_G specified on G. After transformation step 1 (§ 5.2.2), we already obtained the ordinary digraph, which is $\mathbf{T} \in G_1$, and a part of the CCH hypergraph, the latter being number of stop hyperedges contained in $\mathbf{E} \in G_1$. In transformation step 2 (§ 5.2.4), we need to create more hyperedges that represent the desired model components in order to complete the CCH hypergraph. This hypergraph contains Info. Types 6 and 7 (see Figure 5.7).

Since the other hyperedges have compositions that are not as simple as that of the stop hyperedges or as the point composites, we defined graph patterns to search for the occurrences of these composites in G_1. We observed recurrence of some small graph patterns in lager patterns. In order to simplify and reuse the graph patterns and the corresponding search algorithms, we used graph pattern composition (recursive definition and incremental search) in the transformation of the CCH hypergraph.

5.2.3.1 Choosing Graph Patterns and Pattern Composites

The rail infrastructure has a number of characteristics in its geometric (rail track) composition. Three basic composites are the FPs, TPs, and CPs discussed in § 5.2.2.3. They are used as basic units to define the graph patterns in the GPM in transformation step 2 (§ 5.2.4).

There are many different rail infrastructure layouts. Some have similarity in their geometric arrangement but some do not. We shall, however, be able to describe a variety of layouts with a limited number of pattern definitions. How to choose the scope and define these patterns? We use the similar method as in defining the information types and dependencies discussed in § 5.2.1.2. We look at what composites ought to be needed, write down a number of candidates and their relations, consult experts and literature, review them and so forth, in order to complete the information.

Figure 5.12 shows the rail track composites we choose to define[28]. The gray boxes denote different types of hyperedges that shall be specified for the composites (§ 5.2.4). Each (m,n)-edge will be later transformed into a (coupled) model component (§ 5.3). Note the not each type of the rail composite indeed needs a pattern. Types 6∼8, e.g., are simply one-to-one mappings from the points, and Type 9 is so general that it is hardly a pattern; we will explain them in § 5.2.4.

The information about the FP, TP and CP (in the center of the figure) is contained in $\mathbf{P_F}, \mathbf{P_T}$ and $\mathbf{P_C}$ (i.e., the point locations are indicated by the corresponding track vertices) after transformation step 1 (§ 5.2.2.3). The arrows denote composite (information) dependencies or aggregation relations between the composites. It clearly shows in Figure 5.12 that basically all other composite information depends on the information about point composites, which is expected since the physical connections between the rail tracks are enabled by points. In defining the composites, some composites are reused in larger ones; e.g., the "Y" composite (Type 1) contains one FP, one TP and one CP; the "T" composite (Type 3) contains three "Y" composites, while the butterfly union composite (Type 4) contains four "Y" composites.

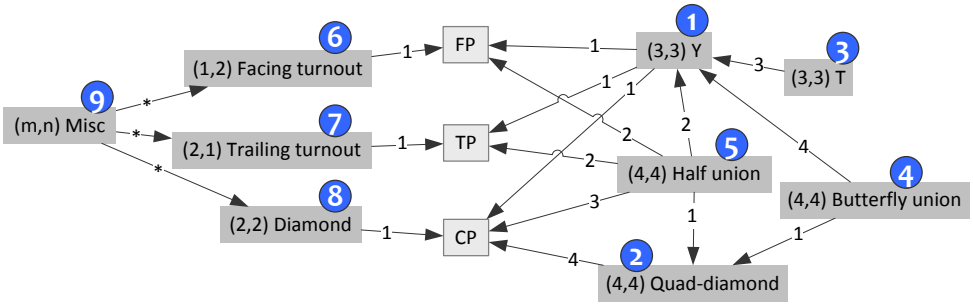


Figure 5.12: Recurring rail composites and their dependencies

[28] They are not complete to define all rail infrastructure layouts but are deemed sufficient to define the infrastructure of HTM light-rail services for our studies.

5.2.3.2 Representing Graph Patterns and Pattern Composites

The main task in transformation step 2 is looking for occurrences of the rail composites in G_1. In order to do so, we need to describe them in graph patterns. In the following, we use the "Y" Composite as an example to explain the graph patterns.

"Y" Composite This composite is a (3,3)-edge representing a common rail arrangement[29], which is already used two times in this thesis; see Example 3.1 (Figure 3.2) and Example 4.5 (Figure 4.20). Its schema is illustrated in Figure 5.13. (The gray lines are the tentacles which are not a part of the composite.) A solid-lined arrow denotes a track vertex. A dot-lined arrow denotes a unique independent path that connects the two corresponding terminal vertices.

In an **independent path** (also called *path graph* or *liner graph*) the internal vertices of the path do not incident edges other than the edges in the path. In other words, if the path has any internal nodes, the in-degree (d^+) and out-degree (d^-) of each internal node are both 1.

The "Y" composite has two unique and independent paths p_a and p_b respectively connecting t_p (in the FP) to t_q (in the CP), and t_r (in the CP) to t_s (in the TP).

Ordered Graph Isomorphism Note that the geometric arrangement of the composite (as the other rail composites used in our study) has to be preserved in GMP. This means that a matched image of the pattern can have, e.g., rotations, but there shall not be mirroring (or flipping) in the image or of the image as a whole. For example, if there exists an "image" that has an "equivalent" of p_a (Figure 5.13) which connects t_p to t_j instead of t_q, then this is *not* a match, which would be considered as a match in a general graph where geometric information is typically not represented (JIANG and BUNKE 1996).

The composites we use belong to a special class of graphs. In graph theory, they are called *ordered graphs* (JIANG and BUNKE 1996, 1999). In an **ordered graph**, the edges

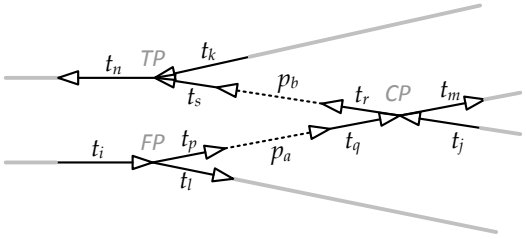


Figure 5.13: "Y" composite (cf. Figures 3.2, 4.20 and 5.10)

[29] This arrangement is sometimes called *double junction*. It is where a double track railway splits into two double track lines. A *double track railway* runs one track in each direction, compared to a *single track railway* where trains in both directions share the same track. In our case, the double track is *right hand running*.

incident to a vertex are uniquely ordered; in a plane graph, e.g., the order can be clockwise or counterclockwise (JIANG and BUNKE 1999). In our case, like in many other applications, the ordering is naturally derived from the underlying geometry of the patterns represented by the graphs (ibid.). An *ordered graph isomorphism* is generally constrained, in which the ordering property is preserved (ibid.).

Following JIANG and BUNKE (ibid.), an *ordered graph* is a triple $G = (V, E, L)$ where (V, E) defines a graph; for each vertex $v \in V$, the edges $(v, v_1), (v, v_2), \cdots, (v, v_k)$ incident to v have a unique order which is represented by a cyclic list $L(v)$.

Two ordered graphs $G = (V, E, L)$ and $G' = (V', E', L')$ are *isomorphic* if there exists an isomorphism f between the two graphs (V, E) and (V', E') such that the order is preserved; that is, if for any vertex $v \in V$, we have $L(v) = \langle (v, v_1), (v, v_2), \cdots, (v, v_k) \rangle$, then $L'(f(v)) = \langle (f(v), f(v_1)), (f(v), f(v_2)), \cdots, (f(v), f(v_k)) \rangle$ holds (ibid.).

JIANG and BUNKE (ibid.) propose an algorithm that can optimally solve the ordered graph isomorphism problem[30] (cf. § 5.1.2.2) in quadratic time, i.e., $O(n^2)$. In our case, the rail composites have more constraints than general ordered graphs: the rail composites are planar and sparse (with specific geometry), and they have bounded vertex degrees and bounded path distances. We therefore choose to take advantage of these properties by designing an algorithm that walks the candidate subgraphs (which therefore only takes linear time) to solve the isomorphism problem. The algorithm is explained in § 5.2.3.3.

Ordering Track Compositions of Points In preparation, we set out to order the composites by first ordering the track compositions of points. Figure 5.14 illustrates the ordering. The tracks around the point center are simply ordered by their directions, viz., entry (*en*) or exit (*ex*). When there are two entry or exit tracks, they must form an angle. We take the non-reflex angle θ (i.e., $0 < \theta < \pi$) as reference, and number the tracks (i.e., two angle sides) counterclockwise[31] as 1 and 2. Note that the geometry of a point naturally imposes unique orderings of the tracks surrounding it.

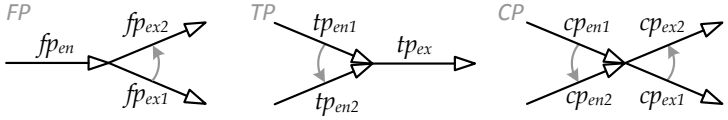


Figure 5.14: Ordering of track compositions of points

[30] JIANG and BUNKE (1999)'s algorithm uses encoding of Eulerian circuits of ordered graphs starting with an edge in an graph. Since the code depends on the choice of the starting edge, $2n$ codes can be generated for some ordered graph of n edges, which can uniquely represent the graph. Ordered graph isomorphism is determined by comparing the codes of two graphs (whether isomorphism exists) and checking the order-preserving property of the two (ibid.).

[31] Since there is always one track on the right side and the other on the left when we rotate the angle θ pointing downwards, we aliased the Nr. 1 track as "right" (R) and the Nr. 2 track as "left" (L).

Representing (Common) Composites with Ordering Using the above orderings, we can define (common[32]) rail composites which would logically have unique orderings because the composites are planar, and the tracks surrounding the points are ordered, and the paths between them (if any) are independent.

The graph pattern defined for the "Y" composite, e.g., contains two independent paths connecting three points[33] in a specific order. The pattern and its illustration are shown in Table 5.1. The (3,3) type implies the number of entry and exit vertices (see § 5.2.2.4) whose mapping relations (in case of a match) are specified. The (constituent) composites and paths specify the matching conditions. In addition, a bound b is specified to constrain the search scope. In our case, this bound is the max search distance[34] of each path.

A GPM algorithm only needs to search for the existence of the points and paths in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$ and $\mathbf{T}$ respectively (see § 5.2.2.3 Figure 5.10) according to the pattern definition. When there is a match, a hyperedge is created to represent the occurrence of the composite by specifying the boundary vertices accordingly. A created hyperedge is recorded in the hyperedge map $\mathbf{E}$ indexed by its entry vertices as discussed in § 5.2.2.4 (Figure 5.11).

Planar composites defined by ordered planar composites and independent paths are also uniquely ordered. The graph pattern defined for the "T" composite, e.g., shown in Table 5.2, contains three "Y" composites connected by six paths in a specific order. The pattern definition has the same form as the previous example, but the definition is

"Y" composite $(3,3)$ e_Y		
Composites		fp, cp, tp
Paths	p_1	fp_{ex2} to cp_{en1}
	p_2	cp_{ex2} to tp_{en1}
Bound		$b \in \mathbb{N}$
Entry	1. e_{Yen1}	fp_{en}
	2. e_{Yen2}	cp_{en2}
	3. e_{Yen3}	tp_{en2}
Exit	1. e_{Yex1}	fp_{ex1}
	2. e_{Yex2}	cp_{ex1}
	3. e_{Yex3}	tp_{ex}

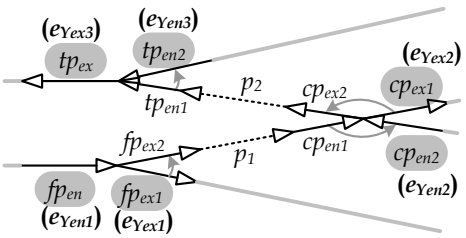


Table 5.1: "Y" composite defined as a graph pattern ϱ_Y

[32] See next paragraph for why they are qualified as common.

[33] Points with ordering means that they are represented by ordered tracks, i.e., $fp = (fp_{en}, fp_{ex1}, fp_{ex2})$, $tp = (tp_{en1}, tp_{en2}, tp_{ex})$, $cp = (cp_{en1}, cp_{en2}, cp_{ex1}, cp_{ex2})$.

[34] This distance means the geometric distance of a path which is the sum of the lengths of all track vertices in the path. We use "distance" to distinguish with the commonly used "length of a path" in graph theory which refers to the number of edges in a path.

"T" composite $(3,3)$ e_T		
Composites		e_{Y1}, e_{Y2}, e_{Y3}
Paths	p_1	e_{Y1ex1} to e_{Y2en3}
	p_2	e_{Y2ex2} to e_{Y1en2}
	p_3	e_{Y2ex1} to e_{Y3en3}
	p_4	e_{Y3ex2} to e_{Y2en2}
	p_5	e_{Y3ex1} to e_{Y1en3}
	p_6	e_{Y1ex2} to e_{Y3en2}
Bound		$b \in \mathbb{N}$
Entry	1. e_{Ten1}	e_{Y1en1}
	2. e_{Ten2}	e_{Y2en1}
	3. e_{Ten3}	e_{Y3en1}
Exit	1. e_{Tex1}	e_{Y1ex3}
	2. e_{Tex2}	e_{Y2ex3}
	3. e_{Tex3}	e_{Y3ex3}

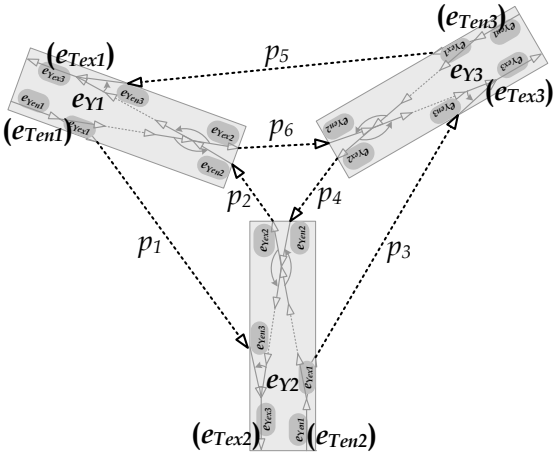


Table 5.2: "T" composite defined as a graph pattern containing three "Y" composites

recursive using other composites. (In GMP, all composites are to be found in **E**). Composites can be defined with both points and composites in the same way. More pattern definitions can be found in § C.2.

Representing Unfixed Composites with Ordering We just presented two examples of *common or regular* rail composites which have "fixed" geometric arrangements. This means their (constituent) composites and paths are known a priori, and as such they can be predefined and matched with some application graph. But not all parts of rail infrastructure can be designed to have common composites. Figure 5.15 shows a practical example. Representing (automatically) this kind of uncommon rail infrastructure

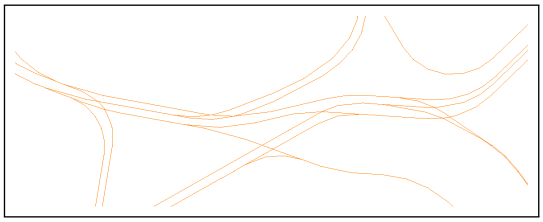


Figure 5.15: Rail infrastructure without common composites

arrangements, we need some composite that can have "unfixed" (constituent) composites *with ordering*. For this purpose, we use a misc (miscellaneous) composite. The *misc composite* is an (m, n) type container. It has hardly a composite pattern since it does not have any specific (constituent) composite. We designed an algorithm that can cluster points that are close enough to one another (given a predefined distance), and each cluster of points is put into a misc composite. The algorithm is explained in § 5.2.4.2.

5.2.3.3 An Algorithm for Composite Isomorphism

In § 5.2.3.2, we discussed two examples of defining common composites with graph patterns (see Tables 5.1 and 5.2). We can define these composite patterns in a general form. A **composite pattern** with boundary definition is $\varrho = (t, C, P, b, f_{en}, f_{ex})$ where

- $t = (m, n),\ m, n \in \mathbb{N}$ is the type of ϱ;
- $C = (c_1, c_2, \cdots, c_w),\ w \in \mathbb{N}$, is a set of pairwise distinct (constituent) composites whose entry and exit vertices are ordered;
- $P = (p_1, p_2, \cdots, p_q),\ q \in \mathbb{N}$, is a set of paths, in which $p_i = (ex_i, en_i),\ i \in [1, q]$ is an independent path that connects an exit vertex of a composite c_{a,ex_x} (i.e., $ex_i = c_{a,ex_x}$) to an entry vertex of another composite c_{b,en_y} (i.e., $en_i = c_{b,en_y}$) where $a \neq b \in [1, w]$;
- $\boldsymbol{b} \in \mathbb{N}$ is a fixed upper bound of the distance of a path p_i denoted by $d(p_i)$, i.e., $d(p_i) \in [0, \boldsymbol{b}]$.
- f_{en} and f_{ex} are the entry and exit vertex mapping functions respectively that maps some entry (or exit) vertices of some constituent composites in C to the entry (or exit) vertices of this (defined) composite.

In a valid composite pattern definition, the size of the entry and exit vertex mapping (by f_{en} and f_{ex}) must match the pattern type (m, n). In addition, all entry and exit vertices of the constituent composites in a pattern must be either connected by a path or mapped to the boundary vertices of the pattern. This means, a pattern ϱ necessarily has the two properties[35] $\sum_{a=1}^{w} m(c_a) = q + m$ and $\sum_{a=1}^{w} n(c_a) = q + n$.

Path Sorting The paths $P \in \varrho$ shall be *sorted* such that a composite c_a that contains $ex_j \in p_j \in P\ (j \in [2, q])$ already "occurred" in a previous path $p \in (p_1, p_2, \cdots, p_{j-1})$ in the sense that c_a contains a terminal vertex of p. This means that the start vertex ex_j of a path p_j (except for the first path p_1) is known at the moment of searching for path p_j. This condition is not necessary for the validity of the composite pattern definition but it does affect positively and significantly the performance of the search algorithm. The path sorting problem can be reduced to the well known *topological sort* (CORMEN et al. 2001). A topological sort is possible if and only if the graph has no directed cycles (ibid.),

[35] For simplicity, we use $m(c)$ and $n(c)$ to denote the number of entry and exit vertices of c respectively, where c can be a composite pattern or a match of a composite pattern.

which is the case of all rail composites we defined. The paths in the "Y" composite and "T" composite definitions, e.g., are topologically sorted (see Tables 5.1 and 5.2).

Given a composite pattern ϱ whose path definition P is topologically sorted, a *composite pattern matching* (CPM) algorithm is defined to search for all isomorphs of ϱ in a host graph $G = (\mathbf{T}, \mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E})$; see § 5.2.2. The pseudo-code of the algorithm is listed in Alg. 5.1. In principle, it does not traverse G per se, but looks for the occurrences of the (constituent) composites $c'_1, c'_2, \cdots, c'_w$ whose candidates are contained in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E}$[36] through walking the independent paths (contained in $\mathbf{T}$) that connect them.

Since $P \in \varrho$ is topologically sorted and the non-match of a path excludes the match of the whole pattern, the vertices that satisfy the conditions of the start vertex ex_1 of the first path p_1 are chosen as candidates to start each pattern search. These candidates are saved in T for iteration (ln. 3)[37]. A candidate vertex t_j is used as the start vertex s of the first search path $p_1 \in P$ (lns. 6, 7).

As mentioned in § 5.2.3.2, the internal vertices of an independent path must satisfy $d^+ = d^- = 1$. In our case, this means that the internal vertices shall *not* appear in the composite maps but only in $\mathbf{T}$. Leaving the search bound $\boldsymbol{b}$ aside, a sufficient condition to consider a vertex $\hat{t}$ to be in a path (including the target vertex) is that it is indexed by an known vertex t in the path (lns. 11, 12). If in addition $\hat{t}$ is not in the composite maps, then we can accumulate the search distance d and continue the path walk (ln. 28). If a known vertex t has no entry in $\mathbf{T}$, then t is literally at the end of the path (with nothing connected to it). In this case, and of course also when the path distance exceeds $\boldsymbol{b}$, the algorithm drops the current search and proceeds with the next candidate $t_j \in T$ (ln. 32).

In walking an independent path p_i given a start vertex s (lns. 11∼30), once a following vertex t appears in the composite maps, it means that there would be a match of p_i if t satisfies the target vertex en_i specification of p_i (i.e., the first two conditions in ln. 14). However, this is *not* necessarily an isomorphism. We need to check whether the found composite $c(t)$, that contains the target vertex t of the (path) image, is pairwise distinct with the other found composites (which are recorded in C'). Note that $C \in \varrho$ is pairwise distinct, hence we have the third condition $c(t) \notin C' \backslash c'_b$ in ln. 14.

If one of the three conditions above is not satisfied (i.e., a non-match) then we can proceed with the next $t_j \in T$ (ln. 24). If these is an isomorphic path match of p_i, the composite $c(t)$ is recorded in C' at the same position as its counterpart in $C \in \varrho$ (ln. 15; note that the index of $c'_b \leftarrow c(t)$ is b as that of $c_b \in C$). In the second case, the algorithm shall proceed with the next path p_{i+1} if there is any path left. Because $P \in \varrho$ is topologically sorted, the composite that contains the start vertex s of p_{i+1} is already in C'. Since the composites in C' are reordered in the same order as in C, we can know s by replacing the containing composite of $ex_{i+1} \in p_{i+1}$, say $c_f \in C$, with its image $c'_f \in C'$ (ln. 20). The path walk given a start vertex s is stated above.

There is an isomorphism if and only if all q paths are found as specified in $P \in \varrho$ (ln. 16). If so, some actions can be performed. In our case, an (m, n)-type hyperedge

[36] We hereinafter use **composite maps** referring to the $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E}$ as a whole.
[37] Line is abbreviated as ln., lines as lns.

Algorithm 5.1 The CPM Algorithm

Require: ϱ

1 $C' \leftarrow (c'_1 \leftarrow \emptyset, c'_2 \leftarrow \emptyset, \cdots, c'_w \leftarrow \emptyset)$ ▷ a set of empty elements the same size as $C \in \varrho$
2 ▷ suppose $ex_1 = c_{a,ex_x} \in p_1$, $c_a \in C$, $a \in [1, w]$
3 $T = (t_1, t_2, \cdots, t_r) \leftarrow ex_x$ of all c from $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$, or $\mathbf{E}$ according to the class of c_a
4 **for** $j = 1 \rightarrow r$ **do** ▷ check all possible candidates $t_j \in T$
5 clear C' ▷ reset C' to empty
6 $s \leftarrow t_j$ ▷ start vertex s of the first search path p_1
7 $c'_a \leftarrow c(s)$ ▷ see ln. 2, cf. ln. 15
8 **for** $i = 1 \rightarrow q$ **do** ▷ check all paths $p_i \in P \in \varrho$
9 $d \leftarrow 0$
10 $t \leftarrow s$
11 **while** t in $\mathbf{T}$ **and** $d < \boldsymbol{b}$ **do** ▷ path search of p_i within bound $\boldsymbol{b} \in \varrho$
12 $t \leftarrow \mathbf{T}(t)$ ▷ $\mathbf{T}$ has an entry $(t, \hat{t})$
13 **if** t in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$, or $\mathbf{E}$ **then**
14 **if** $c(t)$ is in the same class as c_b **and** t is en_y of $c(t)$ ▷ see ln. 15
and $c(t) \notin C' \backslash c'_b$ **then** ▷ a match of p_i
15 $c'_b \leftarrow c(t)$ ▷ suppose $en_i = c_{b,en_y} \in p_i$, $c_b \in C$, $b \in [1, w]$
16 **if** $i = q$ **then** ▷ all q paths matched, i.e., a match of pattern ϱ
17 RECORDMATCH(C', ϱ) ▷ do something related to the match
18 ▷ the loop goes to next t_{j+1} (ln. 4) after ln. 26
19 **else** ▷ suppose $ex_{i+1} = c_{f,ex_x} \in p_{i+1}$, $c_f \in C$
20 $s \leftarrow ex_x$ of $c'_f \in C'$ ▷ start vertex s of the next search path p_{i+1}
21 ▷ the loop goes to p_{i+1} (ln. 8) after ln. 26
22 **end if**
23 **else** ▷ a non match of p_i
24 $i \leftarrow q$ ▷ force the loop to go to next t_{j+1} (ln. 4) after ln. 26
25 **end if**
26 **break**
27 **else**
28 $d \leftarrow d(t) + d$ ▷ accumulate path distance; search continues at ln. 11
29 **end if**
30 **end while**
31 ▷ a non match of p_i because the path ended (with a sink) or bound $\boldsymbol{b}$ is reached
32 $i \leftarrow q$ ▷ force the loop to go to next t_j (ln. 4)
33 **end for**
34 **end for**

- The text following a ▷ symbol is comment.
- A prime sign (′) is used next to the original symbols of the elements in the pattern ϱ, to denote a placeholder for the images of the elements (in case of a match); e.g., C' holds images of elements in C (ln. 1).
- The "class of composite c", e.g., ln. 3, refers to whether c is an *fp*, *tp*, *cp* or a hyperedge composite e.
- The "composite c that contains t" is denoted by $c(t)$, e.g., ln. 14.

e that represents the match is created by RECORDMATCH(C', ϱ) (ln. 17), and the hyperedge map $\mathbf{E}$ has to be updated correspondingly. The CPM algorithm continues with the next $t_j \in T$. The match is recorded in $e \in \mathbf{E}$ so that the algorithm only needs to record the most recent match in C'. Before a new search with the next candidate, C' is cleared (ln. 5). The algorithm terminates after it iterates through all the candidates.

Since the host graph and the (planar) composite patterns are ordered, and the paths in the composite patterns are topologically sorted, the CPM algorithm in the worst case takes liner time $O(n)$ to the (edge) size of the host graph, as it will walk the entire host graph, by which the performance is comparable to that of a graph traversal. The average performance is often substantially better than the worst case, because given the rail infrastructure graph and composite patterns we use, the matches and partial matches of patterns rarely spread densely over the whole host graph.

5.2.4 Transformation Step 2

Transformation step 1 produces a directed graph $G_1 = (\mathbf{T}, \mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E})$ with bounded vertex degrees (i.e., the in- or out-degrees are 1 or 2; § 5.2.2). As preparation for applying CPM (§ 5.2.3.3), in this step, the track composites for points (in $\mathbf{P_F}$, $\mathbf{P_T}$, $\mathbf{P_C}$) are first ordered as discussed in § 5.2.3.2. The orderings are saved as labels in the vertices. The rail composite patterns (Figure 5.12, 1∼5) are defined as discussed in § 5.2.3.2, and the paths in each pattern are topologically sorted as discussed in § 5.2.3.3.

5.2.4.1 Rule Application Control

The transformation in this step is composed of eight sub-steps (cf. Figure 5.12), among which (i)∼(v) apply the CPM algorithm (§ 5.2.3.3):

(i) Detect "Y" composites (see § 5.2.3.2 Table 5.1)
(ii) Detect quad-diamond composites (see § C.2 Table C.2)
(iii) Detect "T" composites (see § 5.2.3.2 Table 5.2)
(iv) Detect butterfly unions (see § C.2 Table C.3)
(v) Detect half unions (see § C.2 Table C.4)
(vi) Decompose small "Y" composites
(vii) Transform (facing and trailing) turnouts and diamond composites (see § C.2 Tables C.5∼C.7)
(viii) Detect misc composites (see § C.2 Table C.8)

Partial Order of Sub-Steps The sub-steps are partially ordered such that the required composites in a step are prepared by one or more previous steps. One can find correlations between Figure 5.12 and the order of the sub-steps; e.g., (i) and (ii) are ordered before (iii)∼(v) as the dependencies that are shown in the figure. During the CPM in (i)∼(v), a match is followed by a rewriting (performed by RECORDMATCH(C', ϱ); see

§ 5.2.3.3) where an (m, n)-edge representing the match is created and the corresponding constituent composites are removed from the composite maps.

Transforming matches into hyperedges allows us to simplify the pattern definitions, match routines and to aggregate match results. In literature, works related to pattern/transformation composition, incremental/recursive pattern matching can be found, e.g., in BALOGH and VARRÓ (2006), VARRÓ et al. (2008), ASZTALOS et al. (2011) and BERGMANN et al. (2012). The first two works concern pattern compositions where a pattern may call itself or other patterns recursively; the search plan is based on flattened graph patterns. BERGMANN et al. (2012) studies the computation of transitive closure since it is needed for recursively defined patterns. ASZTALOS et al. (2011) studies the procedure of composing two transformation rules into a new rule; the application of the new rule is equal to the sequential application of the two original rules.

Our approach starts with the matching of smaller graph patterns, i.e., sub-steps (i) and (ii), during which the matches are transformed into intermediate structures (i.e., the hyperedges) based on which the matching of larger (higher-level) graph patterns are performed, i.e., sub-steps (iii) to (v). The pattern definition takes into consideration and takes advantage that the CCH of the MCG is strictly nested (§ 5.1.1) so that a bottom-up approach of GPM can be performed on the rail infrastructure graph. In addition, since the infrastructure graph is sparse and has bounded vertex degrees (and types and orders) surrounding which the patterns occur (§ 5.2.3.2), the search space of the CPM is reduced to the locations of points, the independent paths that connecting them, and their composites (§ 5.2.3.3).

For example, in sub-step (i), detecting "Y" composites[38], *fp, tp, cp* are contained in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$, and the internal vertices of p_1, p_2 are contained in $\mathbf{T}$. After sub-step (i), each matched "Y" composite is transformed into a $(3,3)$-edge of type e_Y and recorded in $\mathbf{E}$. As such, in sub-step (iii), detecting "T" composites, all the hyperedges of type e_Y are the candidates. We only need to search for the paths between the boundary vertices of the "Y" composites without stepping into the "Y" composites.

We use a hyperedge e to represent a match denoted by $e = (C', P', B_{en}, B_{ex})$ where all images of the constituent composites, paths, boundary (entry and exit)vertices are stored according to the pattern definition ϱ. The constituent composites are removed from their original container, i.e., the composite maps. The newly constructed e is indexed in $\mathbf{E}$ by its entry vertices B_{en} (§ 5.2.2.4). Note that since the matched constituent composites in a previous sub-step are removed from their original container, they do not appear in a later sub-step.

In sub-step (ii), detecting quad-diamond composites, the candidates are crossing points. The candidates in sub-step (iv), detecting butterfly unions, are the hyperedges of "Y" composite (type e_Y) and quad-diamond composite (type e_Q). Similarly, the candidates in sub-step (iv), detecting half unions, are "Y" and quad-diamond composites and points.

[38] The CPM algorithm is invoked with the "Y" composite pattern ϱ_Y (i.e., CMP(ϱ_Y), § 5.2.3.3). The bound $\boldsymbol{b}$ is set to be 100 meters.

Automorphism in Composite Patterns An automorphism of a graph G is a graph isomorphism from G to itself (WEISSTEIN 2009). An ordered automorphism is an automorphism in which the ordering property is preserved. For example, each point (see e.g., Figure 5.14) has automorphism but not ordered automorphism. There are (ordered) automorphisms[39] in the composite patterns we defined, viz., the quad-diamond composite ϱ_Q, the "T" composite ϱ_T, and butterfly union ϱ_B (Tables C.2, 5.2, C.3).

How does this matter to CPM? It depends. When a composite pattern ϱ has automorphism, its matches or images certainly have automorphisms. Since each image is ordered, it makes a difference how the image is ordered if this image is used further as a constituent composite in a higher-level composite.

Consider a directed ordered square graph s with its four vertices numbered as $1 \sim 4$ as shown in Figure 5.16. The graph s has automorphism, and each element of the automorphism group $Aut(s) = \{(1,2,3,4), (2,3,4,1), (3,4,1,2), (4,1,2,3)\}$ is isomorphic to another[40]. When s is to be embedded into a larger graph, say s' (Figure 5.16), there are four ($|Aut(s)|$) different ways to position s (as an ordered graph), i.e., with edges $(1,2)$, $(2,3)$, $(3,4)$ or $(4,1)$. The question is do we want to deem them as the "same" in ordered graph isomorphism?

Depending on the applications, some may need them being deemed as different while some others not. The GPM in our application shall deem them as same. In this case, automorphism poses a problem when *a composite pattern ϱ contains a (constituent) composite c that has automorphism.* Note that it does not matter whether ϱ itself is automorphic so long that it is not in a higher-level pattern.

For example, in sub-step (iv), the butterfly union ϱ_B (Table C.3) contains a quad-diamond composite ϱ_Q (Table C.2) which is automorphic; and similarly in sub-step (v), the half union ϱ_H (Table C.4) contains ϱ_Q; ϱ_B is automorphic while ϱ_H is not. We are not able to match ϱ_B or ϱ_H correctly unless the automorphism of ϱ_Q is dealt with.

How to deal with this problem? Here are some possible solutions.

1. Use non-automorphic alternatives to replace the automorphic composite c.
2. Define multiple versions of the pattern ϱ.
3. Reposition the matches of the automorphic composite, and reapply the GPM algorithm.

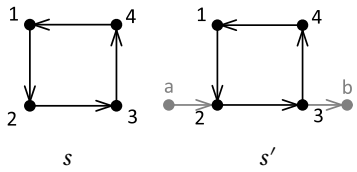


Figure 5.16: Automorphism of a directed ordered square graph s and its embedding s'

[39] It means *ordered automorphisms* since the composite patterns are ordered.
[40] Note that the vertices has *rotations*.

4. Define GPM algorithms that can deal with automorphisms of the composites, e.g., the GPM algorithm can reposition the automorphic composites.

№ 1 is a way around the problem. In defining ϱ, we can avert an automorphic composite c by, e.g., flattening it completely or decomposing it into smaller non-automorphic composites.

№ 2 and № 3 have similar underlying concept, which is to make use of the whole automorphism group. In № 2, this means that each element in the automorphism group of c, i.e., in $Aut(c)$, is used once to define ϱ. This results in $|Aut(c)|$ versions of ϱ. In № 3, each element in the automorphism group of each image c' of c, i.e., in $Aut(c')$, is tested once for matching. In other words, c' is repositioned for $|Aut(c')|-1$ times. Note that $|Aut(c)| = |Aut(c')|$. In *both* cases, the GPM algorithm needs to run $|Aut(c)|$ times. (We assume that algorithm does not by itself deal with the automorphism of c.) If ϱ has automorphism as well, then the number of running the GPM algorithm can be reduced since the "different" versions of patterns or matches are isomorphic.

№ 4 is a generic solution but expensive in solution development compared with the previous three. There are algorithms for ordered graph automorphism in literature, e.g., JIANG and BUNKE (1999). Once the automorphism group is detected, all its members shall be considered as candidates of match[41].

Since we want a simple solution at the first place for reasons explained in § 5.1.2.2 and § 5.2.3.2, we use solution № 3. Recall that our patterns are planar and ordered. Reposition of the matches of ϱ_Q (Table C.2) simply means rotation (as described in the square graph example, Figure 5.16) of the hyperedges of type e_Q, E_Q, which is to rotate the entry and exit vertices respectively. This is very cheap in computation.

Given the geometry of ϱ_B (Table C.3) and ϱ_H (Table C.4), $e \in E_Q$ shall be rotated once in sub-step (iv) and three times in sub-step (v). For example, the CPM algorithm is first run with ϱ_B, on E_Q (and other hyperedges). Then each element left[42] in E_Q is rotated[43], and the CPM algorithm is run on E_Q again.

This makes the CPM algorithm run two times in sub-step (iv) and four times in sub-step (v), if after each run E_Q is not empty.

Composition and Decomposition in Transformation Pattern composition in transformation allows modelers or designers to make stepwise bottom-up definition and application of transformation rules. A good companion of pattern composition is decomposition, which allows composition definitions to be more flexible. Some composites can be useful as candidates for being a part in some higher-level composites. But once they are disqualified as a match in those higher-level composites, we may not need them as composites by themselves. These composites can be decomposed into lower-level composites, which in turn may be used to compose other composites.

[41] The concept is same as described in № 2 and № 3 but the process is done automatically by the algorithm itself.

[42] The matched elements had been removed.

[43] One rotation changes cyclically the positions of the entry or exit vertices in a hyperedge for one place.

Sub-step (vi) of transformation step 2 is an example of this kind of decomposition. In transformation step 1, points are composed based on track geometry. In sub-steps (i)∼(v), a number of rail infrastructure composites are composed based on common point compositions. Briefly stated in § 5.2.3.2, among the points that do not fit into common compositions, those located in short distances (i.e., for a given bound $\boldsymbol{b}$ with a small value) are clustered and placed into a misc composite. We need to do so because these points are interrelated; they shall be supervised by a same control unit (see § 4.3.4.5 and § 4.3.5.2). It turned out that some "Y" composites, even when not contained by a larger composite (i.e., a "T" composite, a butterfly or half union), do not constitute stand alone crossings (or intersections). Some points in these "Y" composites are closely surrounded by some other points which shall form misc composites. These "Y" composites are therefore decomposed[44] in sub-step (vi) in order to make the constituent points available for further composition of misc composites.

The algorithm designed for composing misc composites (§ 5.2.4.2) operates on hyperedges. Therefore, in sub-step (vii), all points left in $\mathbf{P_F}$, $\mathbf{P_T}$, $\mathbf{P_C}$ are transformed (which is an direct mapping) into hyperedges of type e_F, e_{TR}, e_C (see § C.2 Tables C.5 ∼ C.7), and they are indexed in the hyperedge map $\mathbf{E}$ (see § 5.2.2.4 Figure 5.11). Note that the point composites, like the other composites, are ordered. Sub-step (viii) is an application of the misc composite algorithm.

5.2.4.2 An Algorithm for Misc Composites

The *misc composite finding* (MCF) algorithm shall perform the following: given a set of point composites E_P (i.e., hyperedges of type e_F, e_{TR}, e_C as stated above[45]), return a set of misc composites E_M such that the elements in E_P that are connected by independent paths each of which within a bounded distance $\boldsymbol{b}$, are merged into the same element in E_M where each element in e_M shall be ordered.

The main tasks here are (1) path search, (2) merging and (3) ordering. The search of independent paths with bounded distance is straightforward. It is discussed in § 5.2.3.3. This time, the terminal vertices must be boundary vertices of the point composites. We use DFS[46] to walk the paths (in subgraphs). Merging and ordering are performed along the walk when a qualified path is found. The merge concept is similar to agglomerative clustering with single linkage (MANNING et al. 2008), where the individual point composites are merged by progressively merging misc composites. An element is always merged to another element with ordering.

For effective merging and ordering, we designed the (m, n) misc composite such that it manages its structure and order by itself. A misc composite e_M contains

- a point composite map $\mathbf{E_P}$ in which a set of an arbitrary number of point com-

[44] The rewriting of the decomposition is rather simple which is a reverse of creating e_Y: the hyperedge is removed from $\mathbf{E}$ and the points are put back into $\mathbf{P_F}$, $\mathbf{P_T}$, $\mathbf{P_C}$ respectively.

[45] We hereinafter use e_P instead of e_F, e_{TR} and e_C for simplicity.

[46] Again because DFS can be used to classify the edges (CORMEN et al. 2001); the same as in transformation step 1 (§ 5.2.2).

posites $E_P = E_F \cup E_{TR} \cup E_C$ are indexed independently by their entry vertices[47],

- a set of paths P that connect the points,
- an ordered set of m entry vertices B_{en}, and
- an ordered set of n exit vertices B_{ex}.

The boundary vertices of points in E_P are either the terminal vertices of the paths or the boundary vertices of the containing misc composite. An element of E_P or E_M can be merged into another element of E_M with ordering. Let $e_{M1} = (\mathbf{E_{1P}}, P_1, B_{1en}, B_{1ex})$ and $e_{M2} = (\mathbf{E_{2P}}, P_2, B_{2en}, B_{2ex})$ be two distinct misc composites, where

- $B_{1en} = (u_1, u_2, \cdots, u_{m_1})$, $B_{1ex} = (v_1, v_2, \cdots, v_{n_1})$;
- $B_{2en} = (w_1, w_2, \cdots, w_{m_2})$, $B_{2ex} = (z_1, z_2, \cdots, v_{n_2})$.

Suppose that a path from $v_i \in B_{1ex}$ to $w_j \in B_{2en}$ (Figure 5.17) is explored during a walk. Then e_{M2} can be merged into e_{M1} with ordering by the MERGE function[48], as listed in Alg. 5.2. Because the two terminal vertices of path (v_i, w_j) are no longer boundary vertices in the merge, they are removed from the boundary vertex sets. Each removal splits the respective entry or exit vertex set into two parts which are joined to the two ends of the corresponding set of the other misc composite. This boundary vertex ordering[49] is illustrated in Figure 5.17.

Suppose that the path (v_i, w_j) is from a misc composite e_M to a point composite e_P. Then e_P can be merged into e_M by first merging e_P into an empty misc composite, as listed in Alg. 5.3. Merging a misc composite into a point composite can be performed

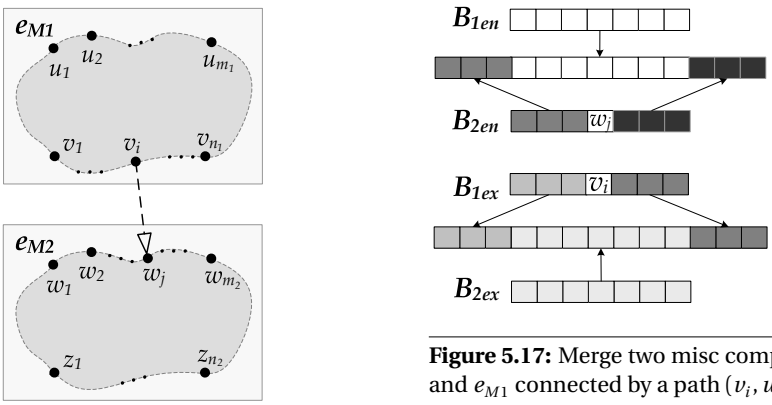


Figure 5.17: Merge two misc composites e_{M1} and e_{M1} connected by a path (v_i, w_j)

[47] See § 5.2.2.4 Figure 5.11.

[48] For generality and readability of the Algorithms, we adapted the their signatures. In our design, these functions are in the misc composite.

[49] Note that it preserves the original ordering which numbers the entry or exit vertices counterclockwise, since the orderings of the point composites are counterclockwise.

Algorithm 5.2 The MERGE Function of e_{M2} to e_{M1}

```
1 function MERGE(e_M1, v_i, w_j, e_M2)
2     E_1P ← E_1P ∪ E_2P
3     B_1en ← (w_1, ··· , w_{j−1}) || B_1en || (w_{j+1}, ··· , w_{m_2})
4     B_1ex ← (v_1, ··· , v_{i−1}) || B_2ex || (v_{i+1}, ··· , v_{n_1})
5     P_1 ← P_1 ∪ P_2 ∪ (v_i, w_j)
6 end function          ▷ The || symbol denotes the concatenation of two ordered sets.
```

in the same manner.

Algorithm 5.3 The MERGE Function of e_P to e_M

```
1  function MERGE(e_M, v_i, w_j, e_P)
2      e'_M ← MERGE(e_P)
3      MERGE(e_M, v_i, w_j, e'_M)
4  end function
5
6  function MERGE(e_P)
7      e_M ← (E_P ← ∅, P ← ∅, B_en ← ∅, B_ex ← ∅)
8      B_en ← entry vertices of e_P
9      B_ex ← exit vertices of e_P
10     for all u ∈ B_en do
11         E_P ← E_P ∪ (u, e_P)
12     end for
13     return e_M
14 end function
```

Suppose that the path (v_i, w_j) is from a misc composite e_M to itself. Then a cycle in e_M is detected, i.e., (v_i, w_j) is a backward path[50]. The cycle can be closed as listed in Alg. 5.4.

Algorithm 5.4 The CLOSECYCLE Function

```
1 function CLOSECYCLE(e_M, v_i, w_j)
2     B_en ← B_en \ w_j
3     B_ex ← B_ex \ v_i
4     P ← P ∪ (v_i, w_j)
5 end function
```

These merge options combined with the DFS walk of the paths surrounding the point composites make the MCF algorithm (Alg. 5.5). As stated earlier, E_P is the set of (hyperedges of) point composites under consideration and $\boldsymbol{b}$ is the bound of path

[50] See, e.g., CORMEN et al. (2001).

Algorithm 5.5 The MCF Algorithm

Require: E_P, $\boldsymbol{b}$
1 $\mathbf{E_M} \leftarrow \emptyset$ ▷ the misc composite map
2 **for** $e_P \in E_P$ **do** ▷ each DFS walk is rooted with an unvisited e_P
3 **if** e_P in $\mathbf{E_M}$ **then**
4 **continue** ▷ go to the next e_P
5 **end if**
6 $EN \leftarrow$ GETNEXTCONNECTEDPOINTENTRY(e_P,$\boldsymbol{b}$) ▷ the entry vertices are ordered
7 **if** EN is not empty **then**
8 $e_M \leftarrow$MERGE(e_P) ▷ a new e_M is created with e_P, Alg. 5.3
9 $\mathbf{E_M} \leftarrow \mathbf{E_M} \cup (e_P, e_M)$
10 **for** $en \in EN$ **do**
11 **if** $en \neq \emptyset$ **then** ▷ the entry vertex en of a connected point composite
12 $i \leftarrow$ index of en in EN ▷ the order of en
13 $ex \leftarrow B_{ex,i} \in e_P$ ▷ the corresponding exit vertex ex of e_P
14 WALKTREE(e_M, ex, en) ▷ Alg. 5.6
15 **end if**
16 **end for**
17 **end if**
18 **end for**

distance[51]. We use a misc composite map $\mathbf{E_M}$ to keep traces of merging (ln. 1). This map contains a set of (hyperedges of) misc composites E_M indexed independently by their constituent point composites $e_P \in e_M \in E_M$, i.e., each entry in E_M has the form (e_P, e_M). At the start of the algorithm, $\mathbf{E_M}$ is empty.

A DFS walk starts with an unvisited $e_P \in E_P$ (lns. 2∼6). The GETNEXTCONNECTED-POINTENTRY(e_P, $\boldsymbol{b}$) function searches for point composites that are connected (by independent paths within $\boldsymbol{b}$, § 5.2.3.3) to the exit vertices of e_P. It returns an ordered set of the connected entry vertices EN. If an element[52] in EN_{next} is empty, it simply means that a qualified path is not found for the corresponding exit vertex of e_P. If no path is found, we continue with the next e_P (ln. 2); otherwise, e_P "becomes" a misc composite e_M (ln. 8), since the connected point composite will be later merged into it. The newly created e_M is recorded in $\mathbf{E_M}$ indexed by e_P (ln. 9). We shall walk deeper in the current "depth-first tree" along the path(s) indicated by EN. (We are at the tree root.) We do so by passing on the "subgraph" e_M and the terminal vertices of the corresponding path (ex, en) to the WALKTREE function (ln. 14).

The WALKTREE function (Alg. 5.6) walks deeper the tree branch until an unqualified path or a backward or a cross tree path (CORMEN et al. 2001) is reached. Recall that the function is invoked with a misc composite e_M, an exit vertex v of e_M, and an entry

[51] For example, this distance can be set to 10 or 20 meters.
[52] Note that EN_{next} has at most two elements since the number of exit vertices of a point composite is at most two.

Algorithm 5.6 The WALKTREE Function

```
1  function WALKTREE(e_M, v, w)
2      e_P ← the indexed value of w in E                    ▷ the connected point composite
3      if e_P in E_M then                                   ▷ e_P is in a misc composite
4          if e_P in e_M then                               ▷ the misc composite is e_M
5              CLOSECYCLE(e_M, v, w)                        ▷ Alg. 5.4
6          else
7              e'_M ← the indexed value of e_P in E_M
8              for all e ∈ E_P ∈ e'_M do                    ▷ the point composites that are merged into e'_M
9                  E_M ← E_M \ (e, e'_M)
10                 E_M ← E_M ∪ (e, e_M)                     ▷ replace the indexed values
11             end for
12             MERGE(e_M, v, w, e'_M)                       ▷ Alg. 5.2
13         end if
14     else
15         MERGE(e_M, v, w, e_P)                            ▷ Alg. 5.3
16         E_M ← E_M ∪ (e_P, e_M)
17         EN ← GETNEXTCONNECTEDPOINTENTRY(e_P, b)          ▷ the same as Alg. 5.5 ln. 6
18         for en ∈ EN do                                   ▷ the same as Alg. 5.5 lns. 10∼15
19             if en ≠ ∅ then
20                 i ← index of en ∈ EN
21                 ex ← B_{ex,i} ∈ e_P
22                 WALKTREE(e_M, ex, en)
23             end if
24         end for
25     end if
26 end function
```

vertex w of a point composite, where (v, w) is a qualified path.

The point composite e_P that cotaines w can be found in the hyperedge map $\mathbf{E}$ (ln. 2). If e_P is in the misc composite map $\mathbf{E_M}$ (ln. 3), then it is visited and is already merged into a misc composite. And if this misc composite is by chance e_M itself (ln. 4), then (v, w) is a backward path and we use CLOSECYCLE to include this path in e_M. When e_P is in another e'_M, we merge e'_M into e_M (ln. 12). Since e'_M is indexed in $\mathbf{E_M}$ by all its point composites $e \in E_P \in e'_M$, we need to replace all indexed values to e_M (lns. 8∼11). When e_P is not in any misc composite, we merge e_P into e_M and record this merge (lns. 15, 16). Now we can further explore the exit vertices of e_P (ln. 17), the similar situation as at a root vertex, but this time e_P is already merged into a misc composite. The walk goes on as explained until all possible branches are visited.

After all point composites are visited, the MCF algorithm terminates. We update the hyperedge map $\mathbf{E}$ using the misc composite map $\mathbf{E_M}$:

(1) remove from $\mathbf{E}$ the point composites that are merged in to misc composites (i.e.,

$\forall e_P \in e_M \in \mathbf{E_M}$ with their indexes $\forall u \in B_{en} \in e_P$) , and

(2) add to $\mathbf{E}$ all misc composites indexed by their entry vertices (i.e., $\forall e_M \in \mathbf{E_M}$ with their indexes $\forall u \in B_{en} \in e_M$).

Note that there are likely point composites that *remain* in $\mathbf{E}$. These point composites will be transformed individually into model components according to the point types, i.e., e_F, e_{TR} and e_C, in transformation step 3.

Remark The misc composite detection is the last sub-step in transformation step 2. After this step, we obtain graph $G_2 = (\mathbf{T}, \mathbf{E})$, in which $\mathbf{T}$ (unchanged as in G_1) is a map of track vertices, and $\mathbf{E}$ is a map of hyperedges each of which represents a rail composite with a corresponding type (Info. Type 7, cf., Figure 5.7). The information about point composites (Info. Type 6) is contained in the corresponding hyperedges of the containing composites. The (track, track) edges in the independent paths connecting the hyperedges are represented by entries in $\mathbf{T}$ (§ 5.2.2). $\mathbf{T}$ is an ordinary graph that describes vertex-vertex relations at the lowest level of the composition. $\mathbf{E}$ is insofar a sufficient representation of the model composition as a CCH. The transformation from the rail infrastructure data to the MCG is completed at $G_2 = (\mathbf{T}, \mathbf{E})$ (cf., § 5.2.3).

The hyperedges (and the represented composites) are of the following types (cf., § C.2):

- e_S $(1,1)$ stop composite
- e_Y $(3,3)$ "Y" composite
- e_Q $(4,4)$ quad-diamond composite
- e_T $(3,3)$ "T" composite
- e_B $(4,4)$ butterfly composite
- e_H $(4,4)$ half union composite
- e_F $(1,2)$ facing turnout composite
- e_{TR} $(2,1)$ trailing turnout composite
- e_C $(2,2)$ diamond composite
- e_M (m,n) misc composite

Each hyperedge (i.e., composite) type has a corresponding model component (type) in the LIBROS library (see, e.g., § 4.3.2 Figure 4.7 `InfraComponent`). During model instantiation in the next step, each hyperedge $e \in \mathbf{E}$ can be transformed into a coupled model component instance according to its type.

5.3 Model Instantiation

In this step, we shall instantiate a simulation model G_3 which is a `TopLevelModel` (see, e.g., § 4.3.2 and § 4.3.5.2) using the LIBROS model components (§ 4.3.4 and § 4.3.5).

Through the last two transformation steps (§ 5.2.2 and § 5.2.4), starting from the infrastructure data G_0, we obtain G_2, which is a MCG that contains sufficient information about the model structure and composition.

For model instantiation, besides the information in G_2, we also need other information about model configuration. The latter information includes the setting of the model parameter values and the initial values of model variables (§ 3.2.3). Note that this is not discussed in § 5.2.1.2 since it does not concern model CCH.

Additional data sources are used for LIBROS model configuration[53]. They need to be prepared such that the data issues related to syntactic consistency and mapping consistency (i.e., criteria № 5, 8 as discussed in § 3.3.3) are dealt with before model instantiation[54]. From now on, we assume that these date sources are transformed into appropriate unit and structure, and they are indexed with identifiers that are consistent with the identifiers[55] contained in G_2.

5.3.1 On Instantiation of LIBROS Models

The instantiation (or generation)[56] of the LIBROS models in this step is completed along with one DFS graph traversal of $G_2 = (\mathbf{T}, \mathbf{E})$. The traversal, however, does not walk deeper into the hyperedges $e \in \mathbf{E}$ except for the boundary vertices. Once reached an entry vertex of a hyperedge, the walk "exits" the hyperedge and goes on further as usual. In this sense, the hyperedges are treated like vertices in the walk.

Along the walk, model component instances are generated and configured. Each vertex or hyperedge is transformed into a model component according to its type. The vertices in $\mathbf{T}$ are only of type source or track; the former being a coupled model (§ 4.3.5.4) and the latter being atomic (§ 4.3.4). Each hyperedge corresponds to a coupled model (§ 4.3.5.2). The ordinary edges, i.e., the (vertex, vertex) pairs in $\mathbf{T}$, indicate the coupling relations between the model components. Model configuration in our case takes place only at the elementary model level (i.e., in the atomic models).

5.3.1.1 Model Instantiation of Rail Infrastructure Elements

A vertex in $\mathbf{T}$ is transformed into a `Source` (§ 4.3.5.4) or a `TrackSegment` (§ 4.3.4). The latter is an atomic RIE model. One can notice that the other RIE models discussed in § 4.3.4 (see, e.g., Figures 4.17 and 4.24) are not represented by individual vertices in $\mathbf{T}$.

[53] We used the following data for LIBROS model configuration. They are provided by HTM.
(1) Timetables that schedule the services.
(2) The routes of the service lines.
(3) Vehicle types for the service lines.
(4) Service line transformation, i.e., the locations where a service line changes to another.

[54] Common approaches, such as data type and format conversion, data merging and mapping tables, are used for the preparation. The implementations of the approaches are domain specific. Therefore we will not discuss the details.

[55] Two types of identifiers appear in G_2: the identifiers for stops and the identifiers for switches.

[56] We use instantiation and generation interchangeably hereinafter in this chapter since they mean the same in the context of transformation step 3.

The 3S models, e.g, are represented by the vertex combinations that are contained in the hyperedges in **E**. They are therefore generated when the hyperedges are generated (see § 5.3.1.2); so are the control units.

The RIE models are defined with fixed structures (including port settings). Hence their instantiation is rather straightforward. Each (atomic or coupled) model instantiation shall specify the parent model of that model. For example, when G_2 is (to be) transformed into a `TopLevelModel` M_{top}, any model that is placed directly under the `TopLevelModel` has M_{top} as the parent.

The configuration of a `TrackSegment` includes its length, radius (if applicable) and speed limit (cf., § C.1) while the 3S models do not have lengths (i.e., length = 0). In addition, for the purpose of animation, a model image is generated for each (atomic) model instance that is to be animated, e.g., we do not animate control units (§ 4.5, Figure 4.34). After the model generation, the images are passed on to the model image manager (Figure 4.35).

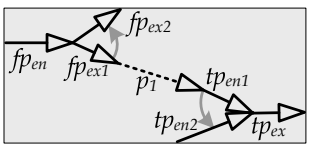
5.3.1.2 Model Instantiation From A Hyperedge

A hyperedge in **E** has specified constituent composites, paths and boundary vertices. The transformation does not "traverse" the hyperedge but instantiates the corresponding model according to the hyperedge in a particular manner. Since many of the composites are intersections, we use a simple misc composite to explain how a coupled model is instantiated from a hyperedge by the *infrastructure component generation* (ICG) algorithm.

Note that the models generated are all *ordered* according to the order of the composites. In particular this means that the start nodes and end nodes (cf., § 4.3.4.1 Figure 4.16) of each model are ordered respectively so that they can be matched with the ordered composites.

Example 5.1 *Transform a* $(2,2)$ *misc composite* e_M *into an infrastructure model* M*. Let* e_M *(and its specification) be as shown in Figure 5.18.*

Given the $(2,2)$-edge e_M, the ICG first creates a (coupled) model M of type `MiscCrossing`[57] as following:



Composites		fp, tp
Paths	p_1	fp_{ex1} to tp_{en1}
Entry	1. $e_{M\,en1}$	tp_{en2}
	2. $e_{M\,en2}$	fp_{en}
Exit	1. $e_{M\,ex1}$	tp_{ex}
	2. $e_{M\,ex2}$	fp_{ex2}

Figure 5.18: A $(2,2)$ misc component e_M

[57] It is an `InfraComponent`, cf., § 4.3.2 Figure 4.7.

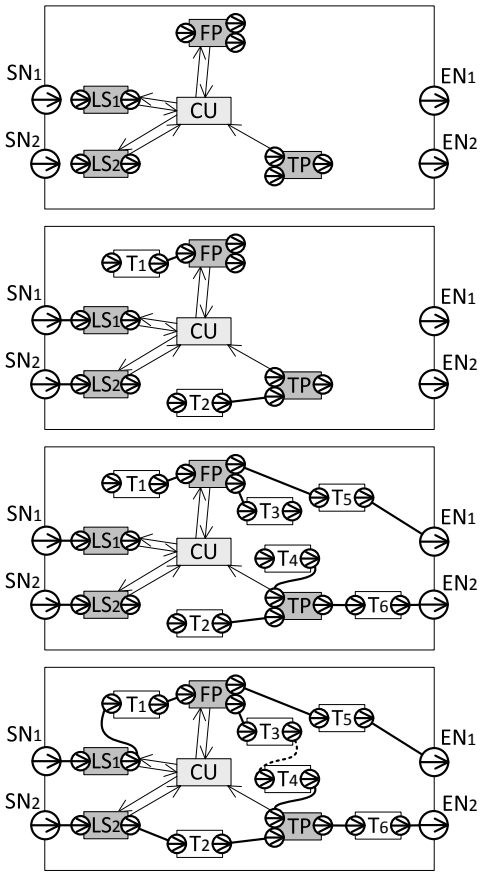
(1) M is initialized with two start nodes SN_1, SN_2 and two end nodes EN_1, EN_2.

(2) Two lineside signals LS_1, LS_2 (cf., § 4.3.4 Figure 4.17) are added, one at each entry (i.e., start node) of the intersection (see the explanation of *Que* below).

(3) A facing point *FP* and a trailing point *TP* (cf., § 4.3.4 Figure 4.17) are added to M.

(4) A control unit *CU* is added to M, with which LS_1, LS_2, *FP* and *TP* are coupled (cf., § 4.3.4.5 Figure 4.20).

This is illustrated in Figure 5.19 (A). For the convenience of infrastructure coupling (cf., § 4.3.5.1 Figure 4.23), a queue like structure *Que* is used to hold successive (1,1) infrastructure models[58] (which together form an independent path) so that they can be later coupled together at once. In a coupled infrastructure model I_C, each start node of



(A) Initialization with start and end nodes; add points, lineside signals and a control unit.
Que status:
$Que_{SN1} = (LS_1)$
$Que_{SN2} = (LS_2)$

(B) Generate track segments in the entry paths and add to entry path *Que*s; couple the start and end nodes of each entry path *Que* to their corresponding attachment nodes.
Que status:
$Que_{SN1} = (LS_1, T_1)$
$Que_{SN2} = (LS_2, T_2)$

(C) Generate track segments in the internal and exit paths and add to *Que*s; couple the start and end nodes of each path *Que*.
Que status:
$Que_{SN1} = (LS_1, T_1)$
$Que_{SN2} = (LS_2, T_2)$
$Que_{FP1} = (T_3, \cdots, T_4)$
$Que_{FP2} = (T_5)$
$Que_{TP} = (T_6)$

(D) Couple the models in the *Que*s.
Que status: the same as above.

Figure 5.19: Steps of generating a coupled infrastructure model M – Example 5.1 (cf., Figure 5.18)

[58] They are components with prefix `Simple-`, § 4.3.2).

I_C and each end node of a non-$(1,1)$ component in I_C is associated with a *Que*. It means that there are five *Que*s in M: two for its two start nodes, two for the two end nodes of *FP*, and one for the end nodes of *TP*. LS_1 is added to Que_{SN1}, and LS_2 is added to Que_{SN2} since signals are to guard the inflow traffic of the intersection.

Next, the ICG generates one track segment (cf., § 4.3.4 Figure 4.17) for each entry vertex of the hyperedge e_M, and adds them into the corresponding start nodes' *Que*s. If an entry vertex of e_M is not an entry vertex of a point component in e_M but holds a path to the latter, then all (models of the) vertices in the path[59] (including the target vertex) are added into the corresponding *Que* (Info. Type 8, cf., Figure 5.7). (Hence the start nodes' *Que*s are also called entry path *Que*s.) To end this step, the (first) start node and the (last) end node (of the models) of each entry path *Que* are coupled to the corresponding attachment nodes. For example, Figure 5.19 (B), since fp_{en} is the first entry vertex of e_M (and the entry vertex of *fp*), a `TrackSegment` T_1 is generated and added to Que_{SN1}. The start node of LS_1 (being the first element in Que_{SN1}) and the end node of T_1 (being the last element in Que_{SN1}) are coupled to SN_1 and the start node of *FP* correspondingly.

The next step, Figure 5.19 (C), is similar to the previous one, but the ICG generates one track segment for each exit vertex of the point composites of the hyperedge e_M and add them to the corresponding *Que*s. Since the exist vertex may hold a path to an entry vertex of another point composite, or to an exit vertex of e_M (or be an exist vertex of e_M itself), these *Que*s are also called internal path *Que*s or exit path *Que*s correspondingly. After these *Que*s are completed, their start and end nodes are coupled to the corresponding attachment nodes as in the previous step.

In the last step, the $(1,1)$ infrastructure models in each (entry, internal or exit path) *Que* are coupled sequentially to one another, i.e., the end node of a previous model is coupled to the start node of a next model. The infrastructure model generation of the hyperedge completes at this step; Figure 5.19 (D).

Remark Note that the shaded model components in Figure 5.19 (i.e., the lineside signals, points, and control unit) do not have corresponding vertices in the hyperedge. The information contained in the hyperedge is sufficient to indicate the composition of the coupled infrastructure model including these extra model components, by which we obtain Info. Type 11 (cf., Figure 5.7). This means that the hyperedge (composite) is injective homomorphic to the coupled infrastructure model.

The four-step model generation is applicable to all intersection hyperedge composites discussed in § 5.2.3 and § 5.2.4. The generation of a stop model needs to transform one independent path (cf., § 5.2.2.3 Figure 5.9) with a sensor (cf., § 4.3.4 Figure 4.17) placed at the entry of the stop model (Info. Type 9). This can be done in the same manner as placing a lineside signal at an entry of an intersection.

[59] The terminal vertices of the path are specified in the hyperedge. The vertices between them are in **T**.

5.3.1.3 Configuration of Control Unit

Each intersection model component in LIBROS contains a control unit (CU). The CU interacts with the detectors and signals in the intersection and supervises the safe operation of the area. In this regard, the use of a check table is discussed in § 4.3.4.5. The check table contains a list of all possible routes in the intersection (Info. Types 10 and 12); Table 4.5 shows an example. This information shall match the infrastructure layout. Since the layout of a misc crossing is not known a priori, we need an algorithm that can configure the routes in the check table automatically according to the layout[60]. Otherwise, the generation of misc crossings is of little use.

Find All Routes in An Intersection A route, denoted by $r = (s, \rho_1, \rho_2, \cdots, \rho_i)$, is composed of an entrance signal and the required points (and positions if applicable) along the route in the intersection. For example, the model M (Figure 5.19) generated in Example 5.1 (§ 5.3.1.2) has three routes, which are $r_1 = (LS_1, FP/2)$, $r_2 = (LS_1, FP/1, TP)$, $r_3 = (LS_2, TP)$. The number following *FP* is the position (i.e., ordering) of the facing point required by the route.

Because all possible routes in an intersection are "rooted" from the entries of the intersection, this finding-all-routes problem is essentially a finding-all-paths problem with given start vertices in a directed acyclic graph. We again use DFS for the route finding, but a search does not stop at a visited vertex. This can be optimized but we did not do so, as an intersection has a bounded number of points (and independent paths) so that the performance is not a concern. We use the *Que*s (§ 5.3.1.2) as "edges" in the search since they represent the entry, internal and exit paths in the intersection.

A search starts at an entry path *Que* whose first element (which is a lineside signal) is added to a new route, i.e., $r = (s)$. The point that is coupled with the end of the *Que* is added into r, e.g., $r = (s, p_1)$. The search walks deeper to an end node *Que* of the point. In case of a trailing point (TP) or crossing point (CP), the route proceeds only in one direction. But in case of a facing point (FP), the original route becomes two routes, and the order of the end nodes in the FP are added into the corresponding route, e.g., $r = (s, p_1/1)$, $r' = (s, p_1/2)$. The search can walk deeper to either end node *Que* with the corresponding route record. The route search continues in the same manner until it reaches an exit path *Que*, and when all the entry path *Que*s are explored.

Assign Routes to Service Lines Each service line has a preassigned service route (from terminal to terminal). This data is provided by HTM (fn. 53 p. 163). The data specifies the direction (i.e., left or right which corresponds to order 2 or 1) of each service line at each facing point. This information needs to be transformed into the corresponding route of the intersection since the vehicle models would request the corresponding route in the simulation. After this route configuration for service lines, the configuration of a control unit is complete.

[60] The algorithm certainly also works on intersections with fixed layout, i.e., those match the composite patterns (§ 5.2.3.2 and § C.2).

5.3.1.4 Model Instantiation and Setting Up Couplings

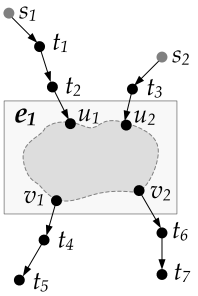
The model instantiation (and couplings) often concern operations of creating and managing nodes and node-couplings[61] that are needed by the infrastructure models in LIBROS. We hence again used the SAM pattern (§ 4.3.2.2, cf., § 4.3.4.4) for the implementation of these operations through `InfraInterface` and `SimpleInfraInterface` (cf., Figure 4.7).

Setting up couplings among models is not as straightforward as it may appear. A coupling relation is a port-to-port relation. Since a model may have more than one start or end nodes, coupling relations often can not be directly translated from model-to-model relations. In Example 5.1 (Figure 5.19), e.g., when `TrackSegment` T_3 or T_5 is to be coupled to *FP*, we need to know the corresponding end node of *FP*. This can be known, e.g., by keeping the order of the end nodes and of the to be coupled `TrackSegments`.

Additionally, the MCG G_2 is injective homomorphic to the generated model G_3 such that a vertex-to-vertex relation in G_2 may represent an indirect model-to-model relation in G_3. This is the case when only one of two connected vertices is a boundary vertex of a hyperedge while the other vertex is not in that hyperedge, so that only one is transformed to be a component of a higher level model. The two corresponding models (of the vertices) as such can not be directly coupled to each other.

Example 5.2 *Consider a simple G_2, as shown in Figure 5.20, with two sources s_1, s_2, one $(2,2)$-edge e_1, and a number of track vertices.*

Suppose the DFS starts at s_1, then it shall first walk through $s_1 \rightarrow t_1 \rightarrow t_2$ because these vertices have entries in **T** but not in **E**. At each step, a model is generated, say S_1, T_1, T_2, and since they do not have multiple start or end nodes, we can successively couple them. At step (t_2, u_1), u_1 has an entry (u_1, e_1) in **E** (i.e., u_1 is an entry vertex of hyperedge e_1), which can be e.g., a $(2,2)$ diamond composite or a misc composite, hence a coupled model, say M_1, is generated according to e_1 (as discussed in § 5.3.1.2). Note



$$\mathbf{T} = \{(s_1, t_1), (t_1, t_2), (t_2, u_1), (s_2, t_3), (t_3, u_2), (v_1, t_4), (t_4, t_5), (v_2, t_6), (t_6, t_7), \cdots\}$$
$$\mathbf{E} = \{(u_1, e_1), (u_2, e_1)\}$$

Figure 5.20: A simple G_2 with two sources s_1, s_2 and one $(2,2)$-edge e_1

[61] A node-coupling is comprised of two port-couplings (on the same two ports) of opposite directions (cf., § 4.3.5.1 Figure 4.23).

that M_1 is the parent model of models of u_1, u_2, v_1, v_2, i.e., U_1, U_2, V_1, V_2 which are all generated at once by generating M_1.

At this point, we need to "couple T_2 to U_1" through a start node of M_1 instead of coupling them directly together. Suppose that M_1 is a (2,2) `MiscCrossing` as shown in Example 5.1 (Figure 5.19 D). Whether T_2 shall be coupled to EN_1 or EN_2 depends on which node U_1 is "coupled to" (through a lineside signal). This information can be known, e.g., through the order of the *Que* that contains U_1[62]. Similarly, when the walk "exists" e_1 and goes to, e.g., $v_1 \rightarrow t_4 \rightarrow t_5$, we use the order of the *Que* that contains V_1 to know the order of the end node of M_1 that shall couple to T_4. Because t_5 is the end of the branch (i.e., it does not have an entry in **T**), we generate a sink model (§ 4.3.5.4), SK_1, and couple it to T_5.

The DFS then starts at the second source s_2 and continues with $s_2 \rightarrow t_3 \rightarrow u_2$. Because there exits an entry (u_2, e_1) in **E** and a model M_1 of e_1 is already generated, we only need to couple T_3 to the corresponding start node of M_1 (as discussed). The walk then goes on to $v_2 \rightarrow t_6 \rightarrow t_7$, couple T_6 to the corresponding end node of M_1, and ends by generating SK_2 after T_7.

Model Map Through traversing a simple example of G_2, we can also observe that some mapping is needed to maintain a relation between G_2 and G_3 in order to trace back which part of G_2 has been transformed into which part of G_3. A DFS in G_2 anyway needs some record to trace the visited vertices. But the model generation (and couplings) also needs a mapping between the vertices (and hyperedges) and the generated models. For example, in traversing the G_2 in Figure 5.20, at step (t_3, u_2), we need to know the counterpart of u_2 in G_3 or whether the model is generated at all in order to have further information about the parent model and the corresponding start port. We therefore use a *model map* **M** of (vertex, model) pairs for the mapping between the vertices in G_2 and their direct counterpart models (not parent models) in G_3. Since the source vertices do not have "back walks", they are not recorded in **M**. Hence **M** has only (track vertex, `TrackSegment`) pairs. These pairs are added into **M** each time a track vertex (including the ones in the hyperedges) is visited and the corresponding `TrackSegment` is generated. The information about the hyperedges (and their corresponding models) can be obtained through the entry vertices.

5.3.2 Transformation Step 3

Through transformation step 2 (§ 5.2.4), the CCH (i.e., the hyperedge map **E**) of the MCG G_2 is defined bottom-up on the ordinary digraph (i.e., the track map **T**). In transformation step 3, as exemplified in § 5.3.1.2 and § 5.3.1.4, the model generation of G_3 is partly bottom-up, as when the graph traversal is at ordinary vertices of **T**, and partly top-down, as when the traversal encounters a hyperedge in **E** based on which an `InfraComponent` is generated.

[62] When the nodes are ordered, the *Que*s are ordered too.

Given $G_2 = (\mathbf{T}, \mathbf{E})$, we use a *model generation algorithm* (MGA) to generate G_3. The pseudo-code of the algorithm is listed in Alg. 5.7. The algorithm is in principle explained by the two examples in § 5.3.1. The model map $\mathbf{M}$ (ln. 1) is created before the model generation. Its entries are added when a `TrackSegment` is generated either during the DFS (ln. 16) or along with the model generation for a hyperedge (ln. 22).

At each DFS walk step, the GENERATEMODELTREE function is passed on with a vertex v in $\mathbf{T}$ and an end node en (ln. 8). The vertex v has a generated model, say V, and it leads the current branch of the tree walk. The end node en is either that of V[63] or an end node of the parent model M of V (i.e., V is in M) to which V is connected with. In both cases, en is the end node that the next generated model shall be coupled with. We may encounter one of the four situations for a walk step (see also Example 5.2 in 5.3.1.4):

(1) When v does not have a descendant vertex, a sink is generated (lns. 10∼12).

(2) When v has a descendant vertex t that is not in a hyperedge[64], a track segment is generated and the walk goes one step further (lns. 15∼18).

(3) When v has a descendant vertex t in a hyperedge e, and a model for t is not generated, a model M for e is generated and the walk goes one step further to one of the branches, (lns. 21∼31).

(4) When v has a descendant vertex t in a hyperedge e, and a model for t is generated, there is a back walk (lns. 33∼35).

Note that v can have at most one descendant vertex since all the point composites are in hyperedges where the DFS does not step into. Model generation for hyperedges (ln. 22) is explained in § 5.3.1.2. In case of an intersection model, the configuration of the control unit is included in the model generation. The MGA terminates when all the DFS trees rooted from the source vertices are explored.

5.4 Model Generator

In LIBROS, there is a `ModelGeneator` component that implements the steps of model transformation (steps 1 and 2) and instantiation (step 3) as discussed in § 5.2 and § 5.3. It is placed in a package together with other model generation components such as different data readers, containers and rail composite patterns and definitions. The main tasks performed by the model generator (Figure 5.21) can be summarized as following.

1. Read CAD data into G_0 (see § 5.2.1.1)[65]. The CAD entities are the vertices in G_0. The entity types and descriptions are the vertex types and attributes.
2. Transformation step 1. Build a directed graph $G_1 = (\mathbf{T}, \mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}, \mathbf{E})$ where track compositions of points (in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$) and stop composites e_S (in $\mathbf{E}$) are detected.

[63] A source or track model can have only one end node.

[64] The only chance for a back walk is when t is in a hyperedge, i.e., a vertex that is not in a hyperedge can not be revisited.

[65] A Java Ycad library (sourceforge.net/projects/ycad) is used to read AutoCAD DXF files.

Algorithm 5.7 The MGA Algorithm

Require: **T**, **E**

1 **M** ← ∅ ▷ the model map, see § 5.3.1.4
2 **for all** s **do** in source vertices in **T** ▷ each DFS walk is rooted with a source vertex s
3 S ← generate a `Source` instance for s
4 en ← the end node of S
5 GENERATEMODELTREE(s, en) ▷ see ln 8
6 **end for**
7
8 **function** GENERATEMODELTREE(v, en) ▷ cf., § 5.3.1.4
9 t ← **T**(v) ▷ check whether **T** has an entry (v, t)
10 **if** $t = \emptyset$ **then** ▷ nothing is connected to v
11 SK ← generate a `Sink` instance
12 couple en and the start node of SK ▷ the tree walk ends at this branch
13 **else**
14 e ← **E**(t) ▷ check whether **E** has an entry (t, e)
15 **if** $e = \emptyset$ **then** ▷ t is not an entry vertex of an hyperedge
16 T ← generate a `TrackSegment` instance for t
17 couple en and the start node of T
18 GENERATEMODELTREE(t, the end node of T) ▷ the walk goes to next vertex
19 **else**
20 T ← **M**(t) ▷ check whether t (hence e) has a model generated
21 **if** $T = \emptyset$ **then** ▷ generate a model M for hyperedge e; cf., § 5.3.1.2
22 M ← generate an `InfraComponent` instance according to the type of e
23 T ← **M**(t) ▷ the model T is newly generated along with M
24 sn ← the start node to which T is connected in M
25 couple en and sn
26 **for all** $i = 1 \rightarrow$ the size of the end nodes of M **do**
27 v ← the i-th exit vertex of e
28 en ← the i-th end node of M
29 ▷ assume exit vertices and end nodes are simply ordered with indexes
30 GENERATEMODELTREE(v, en) ▷ the walk goes to one of the branches
31 **end for**
32 **else** ▷ a model for the hyperedge e is already generated
33 M ← parent model of T
34 sn ← the start node to which T is connected in M
35 couple en and sn ▷ a back walk: the walk ends at this tree branch
36 **end if**
37 **end if**
38 **end if**
39 **end function**

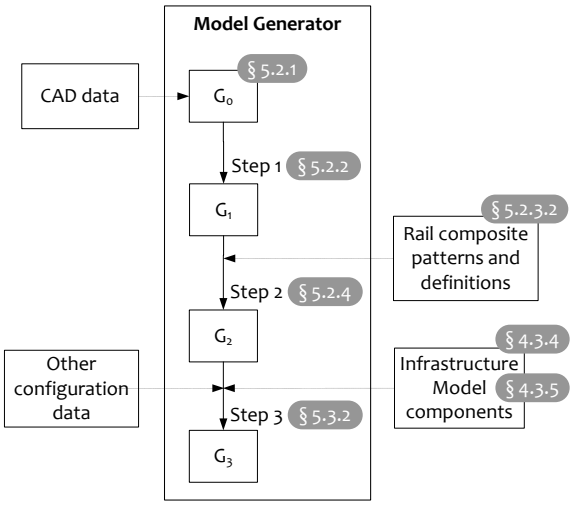


Figure 5.21: Model generator

(a) Partition the vertices into three sets, viz., sources, tracks and stops (cf., § C.1).

(b) "Traverse" G_0 based on the geometrical connectedness of the vertices in order to create directed edges (in $\mathbf{T}$), detect locations of points and create stop hyperedges (based on stop length).

The transformation basically relies on the geometry and the geometrical relation of the CAD entities, particularly those of the track entities, e.g., their positions, connectedness and angles of connectedness.

3. Transformation step 2. Build a MCG $G_2 = (\mathbf{T}, \mathbf{E})$ where $\mathbf{T}$ remains the same as in G_1 and $\mathbf{E}$ contains more infrastructure composites as hyperedges (§ C.2).

   (a) Order track compositions of points (contained in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$). The track vertices that are at one point location are ordered counterclockwise per direction. As such, the whole graph is ordered.
   In sub-steps (b)∼(f), the CPM algorithm looks for ordered subgraph isomorphisms. A match is recorded in a hyperedge and placed in $\mathbf{E}$ indexed by the entry vertices. The corresponding track compositions of points in the composite are removed from $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$.

   (b) Detect "Y" composites e_Y with pattern ϱ_Y. As the other infrastructure composite patterns used in (c)∼(f), ϱ_Y is predefined and ordered.

   (c) Detect quad-diamond composites e_Q with pattern ϱ_Q. A e_Q composite is automorphic.

   (d) Detect "T" composites e_T with pattern ϱ_T (based on e_Y).

(e) Detect butterfly unions e_B with pattern ϱ_B (based on e_Y and e_Q). The CPM algorithm is ran twice with ϱ_B where the e_Q hyperedges are rotated once because of the automorphism.

(f) Detect half unions e_H with pattern ϱ_H (based on e_Y and e_Q and track compositions of points). The CPM algorithm is ran four times with ϱ_H where the e_Q hyperedges are rotated three times because of the automorphism.

(g) Decompose small "Y" composites. The "Y" composites which have points close by are decomposed and the corresponding track compositions are put back into $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$.

(h) Create (facing and trailing) turnouts e_F, e_{TR} and diamond composites e_C. The track compositions of points that are left in $\mathbf{P_F}, \mathbf{P_T}, \mathbf{P_C}$ are transformed individually to hyperedges, viz., points in $\mathbf{P_F}$ to e_F, $\mathbf{P_T}$ to e_{TR}, $\mathbf{P_C}$ to e_C.

(i) Detect misc composites (based on e_F, e_{TR} and e_C). The points that are close to one another are clustered into the same misc composite with ordering.

The transformation starts with G_1 and rewrites it incrementally at each sub-step. These sub-steps are partially ordered. The composite matching (§ 5.2.3.3) or finding (§ 5.2.4.2) algorithm in a sub-step does not traverse the whole graph but walks the independent paths surrounding the candidate constituent composites.

4. Read in the other model configuration data such as timetables and service routes (see p. 163).
5. Transformation step 3. Generate a `TopLevelModel` model G_3 by traversing G_2 and creating infrastructure model components according to the vertex and hyperedge types each of which has a corresponding model component (type) in LIBROS. The components are configured and coupled accordingly. Model images (§ 4.5, Figure 4.34) are as well generated for the purpose of animation.

After these steps, we obtain a simulation model G_3 where the model behavior at the elementary level is pre-specified in the (atomic) model components in the LIBROS library (§ 4), and the model structure is dynamically constructed using the coupled components according to the infrastructure CAD data (§ 5). When empirical data is available, we can also calibrate the model with an automated routine (§ 6).

5

6

Model Calibration

In M&S LITERATURE, the definitions of model calibration are unequivocal[1]. Despite that they all confirm calibration is an iterative process of model adjustment, they are inconsistent in whether the adjustment is made to model parameters only or also to model structure. In this thesis, *model calibration* is an experimental process by which some model parameters (whose values are unknown) are adjusted in a number of simulation experiments in order to match the simulation output (data) with the relevant systems observation or measurement given pre-specified acceptance criteria.

The concept of model validity is briefly discussed in § 2.1.4 and § 4.1.2. It refers to the modeling relation between a model, a source system and an experimental frame (ZEIGLER et al. 2000). The validity of a model is the degree to which the model represents its system counterpart to the extent demanded by the intended model use (ibid.). Although model calibration and validation are two distinct concepts in M&S, they are often conducted together (BANKS et al. 2010).

In general, *model validation* is the overall process of evaluating or testing a model for validity, which includes, i.a., validation of conceptual model, model components and component integration at different model development stages (e.g., KREUTZER 1986,

[1] For example, according to ZEIGLER et al. (2000), calibration is the process by which parameter values are adjusted so that the model best fits the system data. LAW (2007) describes calibration as "…changes are made to the model, somewhat without justification (e.g., some parameter is 'tweaked')." In BANKS et al. (2010), calibration is the iterative process of comparing the model to the real system, making adjustments (or even major changes) to the model, comparing the revised model to reality, making additional adjustments, comparing again, and so on.

BALCI and NANCE 1987, SARGENT 2001, LAW 2007, BALCI 2012, more see § 7). At the stage of model calibration, the model shall be validated and verified for each of the previous stages. As such we can validate the simulation model behavior by comparing it with the system behavior under the same experimental condition (BALCI 1998, ARTHUR and NANCE 2007) — SARGENT (2001) calls this *operational (results) validation* — and reasonably assume that if discrepancy is detected between the two, it is due to inaccurate model parameter configurations. Some parameter values are then adjusted, and the resulting simulation output is again compared with the source system and so forth until the two have a reasonable agreement (MUSSELMAN 1998, LAW 2007). This iterative (operational validation and) calibration process can be automated when the system behavior is recorded in data. Let us call this data empirical data (§ 2.1.2).

This chapter discusses automatic model calibration, which is a necessary component for the completeness of model generation. In literature, parameter search or optimization algorithms for automatic model calibration are extensively discussed[2]. The focus of our discussion is hence placed on an *automatic model calibration procedure* (AMCP), which can work together with the model generator (§ 5.4) as integrated component parts in LIBROS library. We do not aim at defining any specific AMCP, but let users to choose the model calibration elements such as the goodness-of-fit measure or parameter search algorithm according to the problem situation. The model calibration component in LIBROS shall provide a basic underlying procedure that can link together the elements and perform model calibration.

In § 6.1, we first discuss the statistical issues related to output estimates and data comparison based on literature. We then explain the model calibration procedure of LIBROS in § 6.2. Concrete issues about LIBROS calibration experiments are discussed in § 6.3. A test case is presented in § 6.4.

6

6.1 Output Estimation and Comparison

Model parameters can be viewed as model state variables and calibration as the state identification (ZEIGLER et al. 2000). A model calibration procedure defines an iterative experimental process where some parameter configuration in a model is adjusted in each experiment in order to determine whether the model output reasonably agrees with the corresponding empirical data. As such, the *experiments*[3] in one model calibration procedure have the same experimental frame but different parameter configurations. Recall that an experimental frame is a specification of the conditions under which a system is observed or experimented with (ibid., see § 4.1.2).

[2] Parameter search or estimation of complex simulation models is since long proposed as a non-linear optimization problem (TALPAZ et al. 1987, FINLEY et al. 1998). In literature, automated model calibration is often discussed, particularly in terms of parameter search (or optimization) algorithms, e.g., Bayesian method (KENNEDY and O'HAGAN 2001, CACUCI and IONESCU-BUJOR 2010), sequential fitting or iterative search (SOROOSHIAN et al. 1983, ZHANG et al. 2012), genetic algorithm (CHENG et al. 2006, ZAGLAUER and KNOLL 2012), shuffled complex evolution algorithm (YAPO et al. 1996, ECKHARDT et al. 2005), and many others.

[3] An experiment is a process whose outcome is not known with certainty (LAW 2007).

Although AMCPs are often discussed in literature[2], to the best of our knowledge, these discussions refer to equation based or deterministic models as applications. A discrete event model, viewed as a black box, is calibrated essentially in the same way as any other models. Nevertheless, the simulation of a discrete event model that contains (pseudo) stochastic behavior[4] requires output estimation by constructing confidence intervals of the output data in order to obtain a specific precision of the simulation output before the results shall be used for any purpose (LAW 2007, BANKS et al. 2010). Since this is the type of model we use, we need to first estimate simulation output and assure its convergence and then use the results for model calibration.

6.1.1 Output Estimation with Replication Method

In M&S literature (e.g., ALEXOPOULOS and SEILA 1998, LAW 2007, BANKS et al. 2010), many approaches or methods are discussed for constructing the confidence interval of the output data of a simulation experiment. Among those, the *replication method* is the most straightforward and popular (HOAD et al. 2010), where independent replications (or runs)[5] are used to generate (a necessary amount of) output data so as to have a desired precision of the output estimate (LAW 2007, BANKS et al. 2010).

Replications of an experiment are repeated runs of a simulation model (in a same experimental frame) with different streams of random variables (JACOBS 2005, HOAD et al. 2010). Multiple replications are generally required for terminating simulations and can be used for non-terminating simulations (LAW 2007, HOAD et al. 2010, BANKS et al. 2010). Therefore, replications are used for the simulation of LIBROS models[6].

6.1.1.1 Estimation of Mean and Confidence Interval

Let M be a model of a source system S, and Y be an output variable of M. When we simulate M with r replications, we obtain output measures $Y_1, Y_2, \cdots, Y_r$ where $Y_i, i \in [1, r]$ is the output realization of Y from i-th replication[7]. Since the replications are independent, we can reasonably assume that the measures of Y across replications are *independent and identically distributed* (i.i.d).

When a dataset $Y_1, Y_2, \cdots, Y_r$ is i.i.d, the *sample mean* $\overline{Y}(r) = \sum_{i=1}^{r} Y_i / r$ can be used as an unbiased estimator of the *population mean*[8] μ of Y, i.e., $E[\overline{Y}(r)] = \mu$; and sim-

[4] Discrete event simulation models represent stochastic processes through the use of random variables ordered over time (LAW 2007, BANKS et al. 2010).

[5] This means that each replication has the same initial conditions, uses separate sets of different random numbers, and resets the statistical counters (LAW 2007).

[6] The simulations of LIBROS models are terminating.

[7] The data Y_i can be, e.g., the average vehicle driving time from stop A to stop B, or the number of vehicles arrived at a terminal simulated by a LIBROS model in i-th replication. Initialization bias occurs in non-terminating simulations and sometimes also in terminating systems (ROBINSON 2004). The statistics in this section assumes that the initialization bias in the data (if any) already has been reduced to a negligible level, e.g., through intelligent initialization and deletion (BANKS et al. 2010).

[8] In statistical analysis of output data, the mean as a measure of central tendency is a commonly used measure of system performance. For simplicity, we only use estimating mean to exemplify the concept of output estimation. Other useful measures such as quantiles, (expected) proportions can be estimated using

ilarly, the *sample variance* $S^2(r) = \sum_{i=1}^{r}[Y_i - \overline{Y}(r)]^2/(r-1)$ can be used as an unbiased estimator of the *population variance* σ^2, i.e., $E[S^2(r)] = \sigma^2$ (LAW 2007). Because of the random nature in Y, $\overline{Y}(r)$ is random as well with variance $\mathrm{Var}[\overline{Y}(r)]$; an unbiased estimator of this variance is $\widehat{\mathrm{Var}}[\overline{Y}(r)] = S^2(r)/r$ (ibid.). The $100(1-\alpha)$ percent confidence interval for μ is then expressed as such

$$\overline{Y}(r) \pm t_{r-1,1-\alpha/2}\sqrt{\frac{S^2(r)}{r}},\ (0<\alpha<1) \tag{6.1}$$

where $t_{r-1,1-\alpha/2}$ is the upper $1-\alpha/2$ critical point for the Student's t-distribution[9] with $r-1$ degrees of freedom, and $H_{\overline{Y}(r)} = t_{r-1,1-\alpha/2}\sqrt{S^2(r)/r}$ is the confidence interval half-length (see, e.g., ALEXOPOULOS and SEILA 1998, LAW 2007, BANKS et al. 2010).

6.1.1.2 Sequential Procedure with Look Ahead

Suppose that the estimate $\overline{Y}(r)$ has a *relative error*[10] of $\gamma = |\overline{Y}(r)-\mu|/|\mu|$ $(0<\gamma<1)$; we can make (more) replications until the half-length of the confidence interval (in Expression 6.1) divided by $|\overline{Y}(r)|$ is less than or equal to γ, so as to bring the *actual* relative error to be at most $\gamma/(1-\gamma)$ with a probability of approximately $1-\alpha$ (LAW 2007). When the first (mentioned) relative error is adjusted to $\gamma' = \gamma/(1+\gamma)$ to obtain an actual relative error of γ, one obtains the following expression

$$\frac{t_{r-1,1-\alpha/2}\sqrt{\frac{S^2(r)}{r}}}{|\overline{Y}(r)|} \le \frac{\gamma}{1+\gamma} \tag{6.2}$$

which is a necessary and sufficient condition to determine with an $100(1-\alpha)$ percent confidence interval (ibid.):

(1) whether the estimate $\overline{Y}(r)$ of the true mean μ resulted from r replications has a desired precision containing a relative error of γ, and in turn

(2) whether the experiment has made a required number of replications.

Following this principle, the *sequential procedure* presented in ALEXOPOULOS and SEILA (1998) and LAW (2007) first choose an initial number of replications r_0, and more replications are added with Expression 6.2 as the stopping rule, in order to obtain a sufficiently large and smallest possible number of replications. LAW (2007) recommends to use $\gamma \le 0.15$ and $r_0 \ge 10$.

Additional replications, called *look ahead*, are used in HOAD et al. (2010) to avoid potential early (or premature) convergence, i.e., after the condition in Expression 6.2 is satisfied in the sequential procedure, the convergence has to be stable for a number of more successive replications before the the sequential procedure shall terminate.

other formulas; see, e.g., ALEXOPOULOS and SEILA (1998), LAW (2007) and BANKS et al. (2010).

[9] The t-value $t = [\overline{Y}(r)-\mu]/\sqrt{\frac{S^2(r)}{r}}$ (LAW 2007).

[10] $\beta = |\overline{Y}(r)-\mu| > 0$ is called *absolute error* (ibid.).

HOAD et al. (ibid.) recommend a look ahead size $k = 5$ for a total replication number less than 100 and otherwise $k\%$ of the total size.

We therefore use the sequence procedure with look ahead ($\gamma = 0.1, r_0 = 5, k = 5$) for the model calibrator in LIBROS to decide whether to add or stop replications of a calibration experiment.

6.1.1.3 Confidence Interval for Multiple Measures of System Performance

Multiple output variables $Y_1, Y_2, \cdots, Y_n$ of a model M are often used as measures of system performance. In this case, each $Y_j, j \in [1, n]$ is treated as Y discussed afore in the section, and the sequential procedure can be used to construct the confidence interval $I_j = [\overline{Y}_j(r) - H_{\overline{Y}_j(r)}, \overline{Y}_j(r) + H_{\overline{Y}_j(r)}]$ of each output estimate $\overline{Y}_j(r)$ for the true mean μ_j individually (see, e.g., RAATIKAINEN 1993, ALEXOPOULOS and SEILA 1998, LAW 2007). For a chosen α_j for each $\overline{Y}_j(r)$, this means that there is $1-\alpha_j$ probability that μ_j is within the confidence interval I_j, i.e., $P(\mu_j \in I_j) \geq 1-\alpha_j$. However, the probability that all n confidence intervals simultaneously contain their respective true measures, is lower compared to the individual probabilities (see, e.g., LAW 2007). Its lower bound is given by the *Bonferroni Inequality* such $P(\cap_{j=1}^{n} \mu_j \in I_j) \geq 1 - \sum_{j=1}^{n} \alpha_j$ whether or not the I_i's are independent (ibid.).

When n is small (no larger than about 10), then α_i's can be chosen so that $\sum_{j=1}^{n} \alpha_i = \alpha$ where $1-\alpha$ is the desired overall confidence level; but when n is large, the only resource available is to construct the usual 90 or 95 percent confidence intervals but the analysts need to be aware that one or more of these confidence intervals may not contain its true value (ibid.). In applying the sequential procedure, we can evaluate individually the Expression 6.2 for each output variable's estimates across replications[11]. The stopping rule of adding replications applies when all output estimates converge.

6.1.2 Operational Validation through Comparison

Suppose that through the sequential procedure, we obtain a dataset $Y_1, Y_2, \cdots, Y_{r_e}$ from r_e replications of an experiment e, and that the corresponding data from the real system S exists and let $R_1, R_2, \cdots, R_l$ be the dataset. There are different ways to compare them. A number of statistical methods are discussed in literature (e.g., KLEIJNEN 1995, GOLDSMAN and NELSON 1998, BANKS et al. 2010, LAW 2007, SARGENT 2013).

One approach is to use the t-statistic to either construct a confidence interval[12] of the mean difference of the two datasets $\mu_d = \mu_Y - \mu_R$ or to test the null hypothesis $H_0 : \mu_d = 0$ (KLEIJNEN 1995, 1999, LAW 2007). KLEIJNEN (1995) discusses some conditions and potential problems of the hypothesis test due to its stringency, and LAW (2007) recommends the confidence interval over the hypothesis test for the same reasons.

Other often used error (or goodness-of-fit) measures are, e.g, correlation coefficient (KLEIJNEN 1995), mean squares, least squares (VANNI et al. 2011), Kolmogorov-Smirnov

[11] Depending on the practical cases, the α_j's can be different to each other; so do the γ_i's.

[12] When the two datasets do not have paired-up samples, i.e., $r_e \neq l$, the modified two-sample-t confidence interval (i.e., Welch confidence interval) can be used (LAW 2007).

test (HOLLANDER and LIU 2008). Which one or combination to choose depends on the properties of the data and the interest of the measure; reviews can be found, e.g., in SCHUNN and WALLACH (2005) and HOLLANDER and LIU (2008).

6.2 Calibration Procedure

As mentioned earlier, we do not aim at defining any specific AMCP. This means that users shall choose, e.g., the goodness-of-fit measure or parameter search algorithm according to their problem situation, and the model calibration component in LIBROS shall provide the basic underlying procedure that can link together the model calibration elements of users choices.

6.2.1 Basic Elements and Functions

According to SOROOSHIAN et al. (1983), YAPO et al. (1996) and VANNI et al. (2011), the following basic elements are required by an AMCP:

(1) the set of parameters to be included in the calibration,
(2) the parameter space(s) to be searched (typically defined by allowable maximum and minimum values)
(3) the calibration dataset (i.e., empirical data),
(4) the goodness-of-fit measure (i.e., objective function),
(5) the convergence (or acceptance) criteria and stop rule,
(6) the parameter search (or optimization) algorithm.

To carry out an AMCP, the following basic functions need to be performed.

(i) Start a calibration experiment with an initial number of replications (§ 6.1.1). The calibration experiments in one AMCP have the same experimental frame but different parameter configurations, and each experiment is composed of a number of replications that have different streams of random variables (§ 6.3). A new experiment is started if the evaluation of the objective function from the previous experiment is not accepted.
(ii) Add a replication to a calibration experiment if so is required by the sequential procedure (with look ahead; § 6.1.1).
(iii) Read in the calibration dataset. The calibration data provided by HTM is in a database, so we need to define queries to obtain the appropriate dataset.
(iv) Read in the model output dataset. The model state updates are recorded in an output database (§ 4.5), so we need to define queries to obtain the appropriate dataset after each replication.
(v) Perform the statistics defined by the sequential procedure in order to obtain the desired confidence interval of the output estimates across the replications of an

experiment (§ 6.1.1).

(vi) Evaluate the objective function, i.e., compare the output estimates with the calibration data (§ 6.1.2) after the necessary replications of an experiment are completed. Many parameter search or optimization algorithms (packages) includes the evaluation.

(vii) Call the parameter search or optimization algorithm to generate new parameter configuration if the convergence criteria and stop rule are not satisfied.

(viii) The ability to perform multistage calibration.

Multistage Calibration Many calibration procedures apply a stage-by-stage (i.e., multistage) approach where each stage uses a different measure to calibrate a different set of parameters (HOLLANDER and LIU 2008). Examples can be found in HOURDAKIS et al. (2003), DOWLING et al. (2004), VILLA et al. (2004) and CHENG et al. (2005). The ability to perform multistage calibration (viii) means that when the calibration procedure needs to switch from one stage to the next, the initialization conditions of the calibration (or some of them) shall be set anew. Above all, since another set of parameters is to be calibrated, the responsible calibration component shall know the new indexes or references to these parameters. The parameters may have different search spaces or a new objective function. The other conditions that may change include the data queries, the critical values, the allowable relative errors, or even the search or optimization algorithm may be changed.

6.2.2 Procedure Design

Figure 6.1 illustrates the LIBROS model calibration procedure. It consists of activities that perform the functions discusses in § 6.2.1. The sequence of the activities in the procedure is in principle the same as those in classic AMCPs (see, e.g., SOROOSHIAN et al. (1983), YAPO et al. (1996) and HOLLANDER and LIU (2008) and others in fn. 2). The LIBROS AMCP is supplemented with an automated sequential procedure of adding required replications (§ 6.1.1) before each evaluation of the objective function, and with the multistage calibration ability (§ 6.2.1). (Recall that the activities of "evaluate objective function" and "⋯ generate new parameter configuration" are sometimes both included in search or optimization algorithms.)

In the LIBROS library, a number of calibration components are designed to carry out the AMCP and to perform the relevant activities. Figure 6.2 gives an overview of these components and their connection to the LIBROS model which can be a generated (§ 5) or a user defined `LibrosModel`. The two major calibration components are `CalibrationControl` and `LibrosCalibration`. The former controls and executes the overall calibration procedure. The latter deals with concrete calibration issues such as data query and organization, performing statistics and parameter search.

6.2.2.1 Calibration Control

The `CalibrationControl` controls the activity flow of the AMCP. Its functions hence can be seen as the control flow arrows in Figure 6.1, which determine the sequence of the calibration activities (or tasks) and assign these tasks to the (other) corresponding calibration components. These tasks are of two types: those relevant to the addition of experiments and replications, which are assigned to the `CalibrationExperimentHand-`

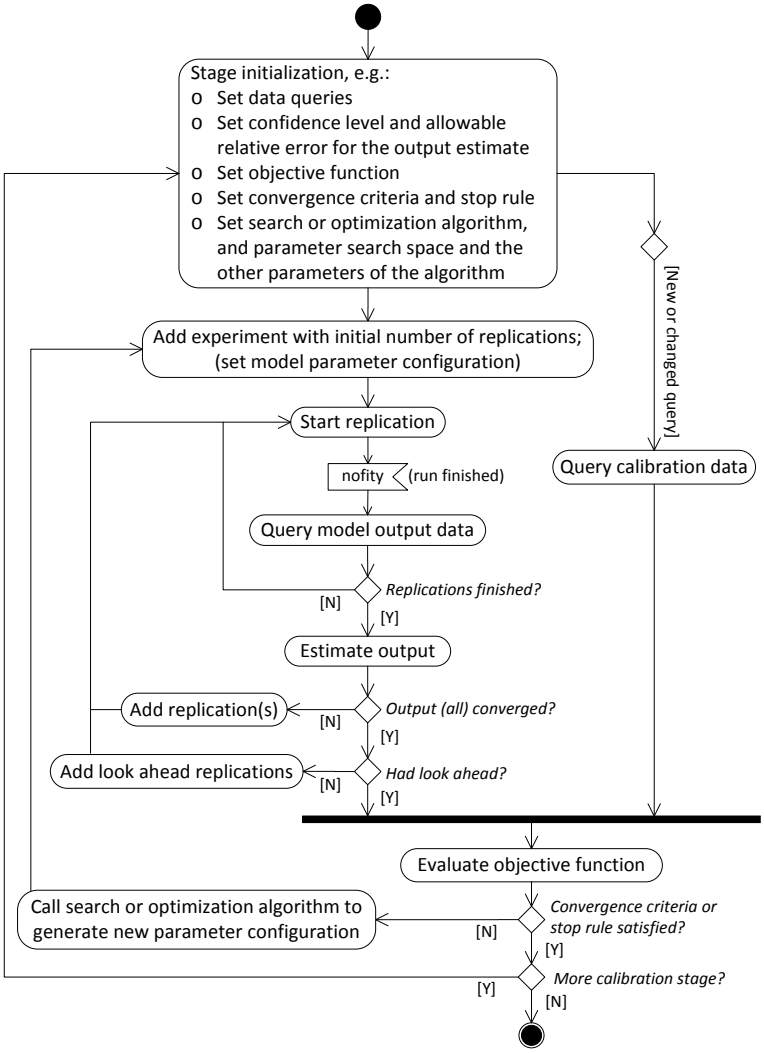


Figure 6.1: LIBROS model calibration procedure

ler, and those relevant to the concrete calibration issues, which are assigned to the LibrosCalibration (§ 6.2.2.2).

Through the CalibrationExperimentHandler, a singleton experimental frame is created for the AMCP (at the first initialization). The ExperimentalFrame, Experiment and Replication are defined as components in the DSOL library (§ A.2); see JACOBS (2005). In the AMCP, an experiment is added to the experimental frame at a stage initialization or when a new parameter configuration is generated for experimentation. Replications are added to a corresponding experiment when the experiment is initialized or when more runs are required by the sequential procedure (§ 6.1.1).

Note that the *master seed* (in the CalibrationExperimentHandler) is used to generate a stream for the random seeds of the replications in an experiment. (Each of the

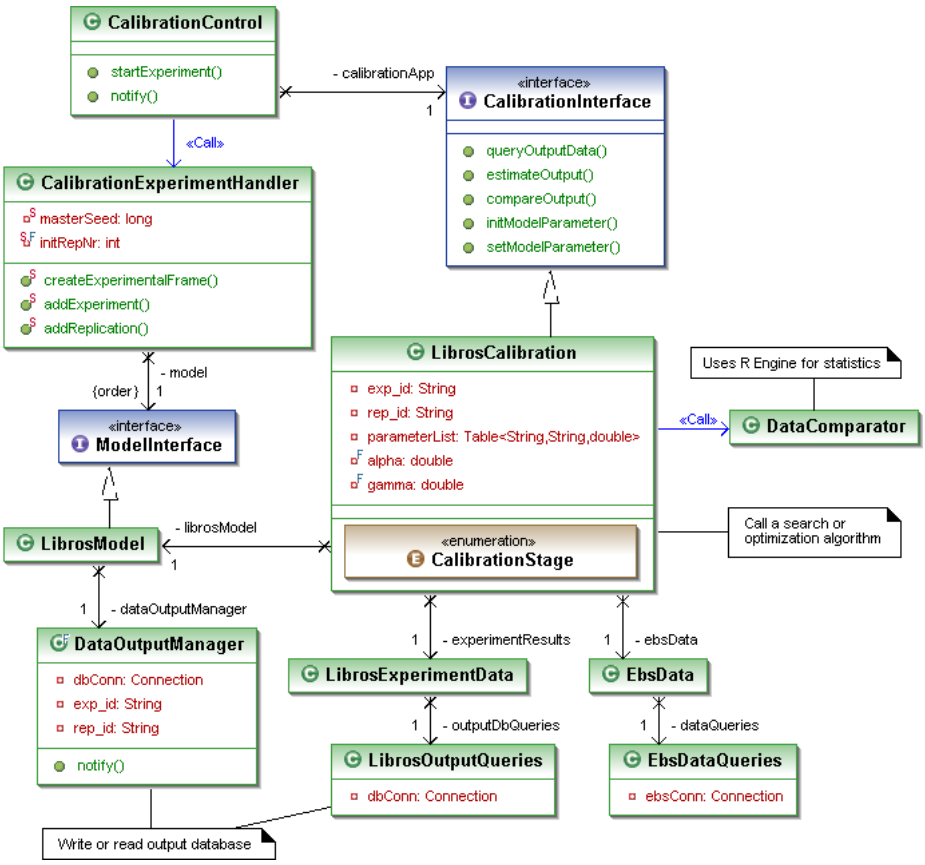


Figure 6.2: Model calibration components connecting with LIBROS model

latter seeds in turn generates a stream of random variables in a replication.) The master seed synchronizes the random number streams (i.e., common random numbers) across different model configurations on a particular replication, which is a simple variance reduction technique (LAW 2007; more variance reduction techniques can be found, e.g., in ibid.).

The `CalibrationExperimentHandler` has access to a `LibrosModel` through the `ModelInterface` in order to create an experiment where a relation between a model and a simulator is set up. Both the `ModelInterface` and the simulator (which is not shown in Figure 6.2) are defined in the DSOL library (§ A.2); see JACOBS (2005).

An experiment (run) is started by the `CalibrationControl`. Replications are ran one after another (by the simulator § A.2), and the output data of interest is queried from the output database (§ 6.2.2.2) after each replication run. In this regard, the publish-subscribe asynchronous communications are again used (cf., § 4.4 and § 4.5), where the `CalibrationControl` subscribes to the "end of replication and experiment events" of the simulator and is subsequently notified of these events (note the `notify` method in the `CalibrationControl`) by the simulator. The data query and statistics after a replication run are performed by the `LibrosCalibration` in order to decide whether a new run is needed. Again, the control flow is decided by the `CalibrationControl`.

6.2.2.2 LIBROS Calibration

The `LibrosCalibration` holds two sets of data queries for a calibration experiment: one for the model output database and one for the database that contains empirical data (sometimes also referred to as calibration data or control data). The latter database used for our calibration experimentation (provided by HTM) is called EBS. The analysts shall choose in advance what and how output and empirical datasets are to be compared and test the correctness of the corresponding queries of the two database systems. Data transformations are often needed in order to obtain the desired comparable data units, types, etc.

The calibration dataset, i.e., $R_1, R_2, \cdots, R_l$, in principle is queried once for one calibration stage. But the analysts may also decide to use the same dataset in different stages. If the statistics in the objective function requires balanced data length (i.e., when output dataset is $Y_1, Y_2, \cdots, Y_{r_e}$, then $r_e = l$), then query for more data points maybe needed since the replication number across experiments may be different.

The output dataset is queried for each replication. Recall that a `DataOutputManager` is used to record model state updates into an output database[13] (§ 4.5). The output data from one AMCP is in the same database file. Hence the query statement must specify the corresponding experiment and replication IDs. The output dataset together with the previous replication datasets in the corresponding experiment are then evaluated for point estimate convergence across replications (§ 6.1.1). And if the sequential procedure (with look ahead) is completed, the datasets from the experiment are compared with the calibration dataset using the given objective function.

[13] We choose to use the Java H2 database (www.h2database.com). It is a light-weight, fast, open source database system suitable to be used for model output logs.

The `LibrosCalibration` gives all the statistical computing tasks to the `DataComparater`. The latter is a component designed to use the R environment[14] for statistics. The task of assigning a new parameter configuration is given to a user specified search or optimization algorithm; an example can be found in § 6.3.2.3. When the `LibrosCalibration` obtains the new configuration, the latter is passed on to the corresponding model parameters in the `LibrosModel` for the next experiment. In the event of ending a calibration stage, the next stage (if any) is started after a stage initialization where some conditions for calibration are set anew. Some examples of these conditions are stated in § 6.2.1.

6.3 Calibration Experiments

Consider that we have a fixed set of data about service infrastructure, routes and timetables. Given these data input and the LIBROS model components as described in § 4.3, the repeated executions of model generation (§ 5) always produce the same models, in the sense that these models have the same structure and parameter configuration, and hence the same model behavior (§ 4.3).

In real urban transport operations, when the service infrastructure, routes and timetables remain the same, the service outcomes seldom remain the same due to many factors, i.a., traffic situation, location, driving behavior (TAHMASSSEBY 2009, VAN OORT 2011). The dynamics of these factors are uncertain in a LIBROS model. However, when measurements of the service outcomes are available, the parameters that represent the unmeasurable dynamic factors can be calibrated.

Many modern transport systems, particularly railway systems, are equipped with sensor and GPS devices that are able to automatically collect data. HTM has abundant data about vehicle trips, travel times, etc., of vehicle daily operations in their EBS (*Exploitatie Beheersingssysteem* in Dutch) database (VELDHOEN 2009).

6.3.1 Case Description

In § 4.3.3, we explained the principle of vehicle movement modeling in which a maximum speed limit is used to set the upper bound of velocity calculation. If the calculation is with strict respect to the *maximum allowable speeds*[15] (or *speed limits*) set by the regulations, then all vehicle models would drive in the same way, which is undesirable for modeling urban light-rail operations as required by HTM. With the suggestions from domain experts, a variable called *maximum speed ratio* (MSR) is used to introduce variations in vehicle movement; we denote it as ϕ. A value of ϕ, let it be φ, is randomly assigned to each vehicle model when it is generated at a source (§ 4.3.5.4).

[14] R is a powerful, open source environment for statistical computing and graphics widely used among statisticians (www.r-project.org). We use a Java/R interface (JRI, www.rforge.net/JRI) to pass data between Java and R, and to access R from Java.

[15] There are three categories of speed restrictions, in the city 35 km/h, city center 15 km/h, outside of city 50 km/h.

When an area has a speed limit $\bar{m}$, then the *effective maximum speed* of a vehicle is determined by $m = \varphi\bar{m}$. For example, when $\varphi = 1$, then the vehicle drives with a maximum speed equal to the speed limit, and when $\varphi = 1.1$ or 0.9 then the vehicle drives with a maximum speed 10 percent higher or lower compared to the speed limit. The experts suggest to use a normal distribution with bounded tails assuming that the "fast" and "slow" drivers are symmetrically distributed.

During simulation, φ is adjusted according to the location of the vehicle. For example, at busier areas where vehicles tend to drive slower and the driving speeds have larger variances, ϕ should have a smaller mean and a larger variance compared to less busy areas. Because successive stop-to-stop (distance) interval is a convenient unit of measure that decomposes a total trip distance (from start to end terminals) into areas, the association of ϕ to different areas is in a stop-to-stop manner. In the model, for each stop-to-stop interval, the value of φ shall be adjusted according to the situation of that area.

Specifically, the value of ϕ is first randomly generated from a *Probability Density Function* (PDF) when a vehicle is generated at a source (§ 4.3.5.4); let us call this random variable ϕ'. Because the normal distribution $\mathcal{N}$ is suggested, we have $\phi' \sim \mathcal{N}(\mu, \sigma^2)$. During simulation, the value of ϕ' is adjusted for each stop-to-stop interval in terms of the location and the scale of the PDF. This means that the adjusted value of ϕ' shall have the same z-score (because of the normal distribution) as ϕ'.

Let φ'_j be the generated value of ϕ for vehicle v_j, and $\varphi_{j,p\text{-}q}$ be the (desired) adjusted value for v_j from stop p to stop q. Suppose that the latter value is from $\mathcal{N}(\mu', \sigma'^2)$ where $\mu' = a_{p\text{-}q} + \mu$ and $\sigma' = b_{p\text{-}q}\sigma$, $a_{p\text{-}q}$ being the location parameter and $b_{p\text{-}q}$ being the scale parameter for the trip interval from stop p to stop q, then

$$\begin{aligned} Z(\varphi'_j) &= Z(\varphi_{j,p\text{-}q}) \\ \frac{\varphi'_j - \mu}{\sigma} &= \frac{\varphi_{j,p\text{-}q} - \mu'}{\sigma'} \\ \varphi'_j - \mu &= \frac{\varphi_{j,p\text{-}q} - a_{p\text{-}q} - \mu}{b_{p\text{-}q}} \\ \varphi_{j,p\text{-}q} &= a_{p\text{-}q} + \mu + b_{p\text{-}q}(\varphi'_j - \mu) \end{aligned} \tag{6.3}$$

Thus, we can know $\varphi_{j,p\text{-}q}$ when we know the variables on the right hand side of Eq. 6.3. Good values of these variables can be approximated through calibration.

6.3.2 Two Stage Calibration

Recall that variable ϕ defines a vehicle's effective maximum speed $m = \varphi\bar{m}$ on the speed limits (set by the regulations) whose values are known. It is reasonable to suspect that $\bar{m}$ is positively correlated with a vehicle's speed or trip time. Hence, we can compare, e.g, the corresponding stop-to-stop speeds from the model and from the empirical data, and attempt to adjust the values of ϕ (or more specifically the values of $a_{p\text{-}q}$, $b_{p\text{-}q}$, μ and σ) using the results of the comparison.

Suppose that there are n stop-to-stop trip intervals in a LIBROS model, and the i-th stop-to-stop speed from the model and from the EBS data are respectively Y_i and $R_i, \forall i \in [1,n]$. Let a_i and b_i be the i-th location and scale parameters in the model.

A two-stage procedure is designed for the calibration. At the first stage, μ and σ are calibrated, and a_i and b_i are fixed with $a_i = 0$ and $b_i = 1$. This means that for each generated vehicle model v_j, a value $\varphi'_j \in \phi' \sim \mathcal{N}(\mu, \sigma^2)$ is assigned and this value is not adjusted at the stop-to-stop intervals. After each simulation experiment, the pairs of Y_i and R_i are compared (§ 6.3.2.1), and if necessary μ and/or σ are adjusted correspondingly (§ 6.3.2.2 and § 6.3.2.3) for the next experiment.

At the second stage, μ and σ are fixed with the calibrated values, and a_i and b_i are calibrated. At each stop-to-stop interval, $\varphi'_j, \forall j = 1,2,\cdots$, are modified such that $\varphi_{j,i} = a_i + \mu + b_i(\varphi'_j - \mu), \forall i \in [1,n]$ following Eq. 6.3. The same as in the previous stage, after each experiment, the pairs of Y_i and R_i are compared (§ 6.3.2.1). If necessary, the individual a_i and/or b_i are adjusted (§ 6.3.2.2 and § 6.3.2.3).

6.3.2.1 Data Comparison

In the sequential procedure, we use $\alpha_i = 0.05, \gamma = 0.1$, $r_0 = 5$, $k = 5$ (§ 6.1.1). For the comparison of the two datasets Y_i and R_i, their PDFs are examined. In § 6.1.2, we stated that hypothesis test with t-statistic is stringent for comparing simulation output data and system measurement (KLEIJNEN 1995, LAW 2007). They can be particularly restrictive for transport applications where traffic conditions often have large variability (KIM et al. 2005). The confidence interval approach (e.g., paired-t or Welch approach) provides information of the magnitude of the differences between Y_i and R_i (LAW 2007), but is less suitable for an automatic procedure. Examining the PDF of the datasets rather than directly examining each individual observation is a way to avoid the effect of small errors (HOLLANDER and LIU 2008).

KIM et al. (2005) propose nonparametric (i.e., distribution-free) methods since they do not require a priori assumptions about the distribution of the underlying population. The *Kolmogorov-Smirnov* (KS) test is a nonparametric test (CONOVER 1971) used for calibration studies such as in KIM et al. (2005) and LEE and OZBAY (2008, 2009) and found suitable to capture characteristics of the observed travel time data. The KS test is therefore used to compare Y_i and R_i and the p-value of the KS test is used in the objective functions (§ 6.3.2.2).

Use the Right Data Comparable datasets need to be queried from the simulation output and EBS databases. This may seems to be obvious but can be overlooked sometimes. As mentioned earlier, the successive stop-to-stop driving speeds of the simulation are compared with EBS data. In this case, the stop-to-stop trip intervals shall first match. In § 3.3.2, mapping consistency (#5.) is identified as a data quality criterion. When the two compared measures do not have the same identifiers, e.g., the stop IDs in the infrastructure data (or timetables) do not match with those in the EBS data, the corresponding mapping information needs to be prepared in advance.

The time interval of the two datasets is another issue. For example, if we simulate the morning-peak 7:00~9:00 (and use the corresponding timetable entries), we probably want to use calibration data with comparable time interval. When the data is abundant with many repeated measurements, we may wish to select a subset according to the situation of the analysis. For example, the EBS data we have contains over one year daily operational data. Then we can choose, e.g., the data from different seasons to calibrate driving times (with corresponding seasonal timetables) characterized by that season. And similarly, weekend or non-weekend data, peak or non-peak hour data can also be analyzed separately. Analysts should pay attention that the data used for model generation, e.g., the timetable data, also needs to correspond with the calibration data[16].

6.3.2.2 Objective Functions

Two objective functions are used for the two model calibration stages. After an experiment, Y_i and $R_i, \forall i \in [1,n]$ are compared by the KS test (§ 6.3.2.1) to obtain n p-values, i.e., $p_i = ks(Y_i, R_i) \in [0,1]$. In the first stage, since φ'_j is carried by a vehicle model v_j through its whole trip in a simulation run, the objective function f_1 aims at the overall "best match" between the output and EBS datasets.

First Stage Let $P_1 = (p_{1,1}, p_{1,2}, \cdots, p_{1,n})$ and $P_2 = (p_{2,1}, p_{2,2}, \cdots, p_{2,n})$ be two n-tuples of p-values from the KS tests of two calibration experiments. We define a pairwise evaluation function $f_e(p_{1,i}, p_{2,i})$, $i \in [1,n]$ on the corresponding p-values such

$$f_e(p_{1,i}, p_{2,i}) = \begin{cases} 0 & \text{when } p_{1,i} \geq \alpha_p \wedge p_{2,i} \geq \alpha_p \text{ or } p_{1,i} = p_{2,i} = 0 \\ 1 & \text{when } p_{2,i} > p_{1,i} = 0 \text{ or } p_{2,i} \geq \alpha_p > p_{1,i} > 0 \\ -1 & \text{when } p_{1,i} > p_{2,i} = 0 \text{ or } p_{1,i} \geq \alpha_p > p_{2,i} > 0 \\ f'_e(p_{1,i}, p_{2,i}) & \text{otherwise} \end{cases} \tag{6.4}$$

$$f'_e(x_1, x_2) = \begin{cases} 0 & \text{when } |x_1 - x_2| \leq \bar{d} \\ 1 & \text{when } x_2 - x_1 > \bar{d} \\ -1 & \text{otherwise} \end{cases} \tag{6.5}$$

where α_p is a specified significance level of the individual p-values, and d is a specified accuracy threshold used to directly compare two values. In the test case in § 6.4, these two values are $\alpha_p = 0.05$ and $\bar{d} = 0.001$. Intuitively, value "1" indicates a higher p-value of $p_{2,i}$, value "−1" a lower p-value, and "0" being equal.

Suppose that we are at the first stage k-th experiment where the model is ran with parameter setting $\theta_k = (\mu_k, \sigma_k)$. After this experiment, $P_k = (p_{k,1}, p_{k,2}, \cdots, p_{k,n})$ is evaluated against P_{k-1} from the previous experiment. Let $E_k = (e_{k,1}, e_{k,2}, \cdots, e_{k,n})$ be the

[16] The data, specially sensor data, often need to be preprocessed such as transformation and cleaning. The related issues can be found in relevant literature. The EBS data we use for model calibration are all preprocessed by the company.

evaluation result where $e_{k,i} = f_e(p_{k-1,i}, p_{k,i})$ as defined above. Then the objective function in the first stage is simply defined as $f_1 : \sum_{i=1}^{n} e_{k,i} > 0$ which means that the count of the improved p-values shall be larger than that of the worsened p-values. If this objective holds, then the next experiment for $\theta_{k+1} = (\mu_{k+1}, \sigma_{k+1})$ remains in the same stage, otherwise the calibration switches to the second stage. Note that in the latter case, $\theta_{k-1} = (\mu_{k-1}, \sigma_{k-1})$ are taken as fixed μ and σ values since the latest experiment did not improve the p-values.

Second Stage In the second stage, the location and scale parameters are calibrated on a stop-to-stop basis. The parameter sets $\theta_{k,i} = (a_{k,i}, b_{k,i}), \forall i \in [1, n]$ in each k-th experiment are calibrated independently. The p-values $p_{k,i} \in P_k$ are compared individually with the specified significance level α_p to determine whether the location and scale parameters for the i-th stop-to-stop trip interval shall continue to be calibrated in the next experiment. This means that, as in the first stage, we do not go for an "optimal" but a good enough configuration. After each k-th experiment, we shall find new parameter settings for $\Theta_{k+1} = (\theta_{k+1,1}, \theta_{k+1,2}, \cdots, \theta_{k+1,m} \mid j \in [1, m], p_{k,j} < \alpha_p, p_{k,j} \in P_k)$ for those stop-to-stop speeds that do not have a significant p-value. We hope to improve the $p_{k+1,j}$ value by using the new configuration $(a_{k+1,i}, b_{k+1,i})$. The objective function is hence defined as $f_2 : |\Theta_{k+1}| = 0 \vee k = \bar{k}$ where $\bar{k}$ is a specified value of the maximum number of experiments.

6.3.2.3 Parameter Search Algorithm

A simple parameter search algorithm is designed for calibration. The underlying principle is intuitive, similar to that of the *control variates* method used for variance reduction (Law 2007). We take advantage of the (positive) correlation between the effective maximum speed $m = \phi \bar{m}$ in stop-to-stop trip intervals and the respective driving speed. It is reasonable to suspect that if the value of m is adjusted higher or lower, the driving speed becomes higher or lower correspondingly.

Suppose that Y_i is the i-th stop-to-stop speed from a calibration experiment and R_i is the corresponding EBS data. Because the objective is to match the PDF of Y_i to that of R_i, we can try to bring the location and scale of Y_i towards those of R_i by adjusting the location and scale of ϕ. For example, if the location of Y_i is to the left of R_i, we can adjust the location of ϕ upwards (referring to the x-axis) by a certain amount; and if the scale of Y_i is larger than that of R_i, we can scale down ϕ.

To compare the location and scale of Y_i and R_i, their mean (μ) and standard deviation (σ) can be used because μ and σ are the location and scale parameters for the normal distribution. This does not mean, however, we have to make any distribution assumption, since μ and σ are only used to assess the "direction" of difference between the two densities. At the same time, we suspect that the bandwidth h from *kernel density estimation* (KDE) (Sheather and Jones 1991) of the PDF can also be a good measure for scale parameter differences. Given the nature of KDE, h and σ are certainly correlated. A comparison test of the two is presented in § 6.4.1.

Suppose that after an experiment, the location and scale measures of Y_i (from simulation data) are $L_Y = (l_{Y1}, l_{Y2}, \cdots, l_{Yn})$, $S_Y = (s_{Y1}, s_{Y2}, \cdots, s_{Yn})$. The corresponding R_i (from EBS data) has $L_R = (l_{R1}, l_{R2}, \cdots, l_{Rn})$, $S_R = (s_{R1}, s_{R2}, \cdots, s_{Rn})$. We compute the difference measures $D_L = (d_{l1}, d_{l2}, \cdots, d_{ln})$ and $D_S = (d_{s1}, d_{s2}, \cdots, d_{sn})$ such that $\forall i \in [1, n]$

$$\begin{aligned} d_{li} &= f'_e(l_{Ri}, l_{Yi}) \\ d_{si} &= f'_e(s_{Ri}, s_{Yi}) \end{aligned} \tag{6.6}$$

in which f'_e is as defined in Eq. 6.5 with $\bar{d} = 0.05$. Simply put, $l_{Yi} - l_{Ri}$ and $s_{Yi} - s_{Ri}$ are used to compute the direction of differences[17].

First Stage In the first stage, since we want to adjust the location and scale of $\phi' \sim \mathcal{N}(\mu, \sigma^2)$ to improve the overall p-value, we shall adjust them towards the "direction of the majority". For example, when there are more Y_i's that have too right locations ($d_l = -1$) than too left locations ($d_l = 1$), we shall move μ downwards by a predefined amount (we call this amount *calibration step*); on the other hand, when there are more Y_i's that have correct locations ($d_l = 0$) than lefts and rights, then μ shall remain unchanged.

Let $D = (d_i), i \in [1, n]$ be D_L or D_S. Two subsets of D are defined such that $D_h = (d_p | d_p = 1, d_p \in D)$ and $D_l = (d_q | d_q = -1, d_q \in D)$ which contain elements that have high value "1" or low value "−1" respectively. The direction of the adjustment of μ or σ is defined such

$$f_d(D) = (-y \mid c_y = max\{c_x\}, x, y \in \{-1, 0, 1\}) \tag{6.7}$$

where $c_{-1} = |D_l|$, $c_1 = |D_h|$, $c_0 = |D| - c_h - c_l$. Intuitively, it means that the direction[18] to adjust μ or σ shall be *opposite* to that of the most count of the high, low and equal location or scale parameters. The parameter configuration for the next experiment is then defined as

$$\begin{aligned} \mu_{k+1} &= \mu_k + \Delta\mu\, f_d(D_L) \\ \sigma_{k+1} &= \sigma_k + \Delta\sigma\, f_d(D_S) \end{aligned} \tag{6.8}$$

where $\Delta\mu$ and $\Delta\sigma$ are specified calibration steps for μ and σ. The values used in the test case are $\Delta\mu = 0.05$ and $\Delta\sigma = 0.025$. The initial values of μ and σ are $\mu_1 = 0.8$ and $\sigma_1 = 0.1$ (for the first experiment).

Second Stage In the second stage, the adjustments of μ and σ (or more precisely the adjustments of a_i and b_i) for each stop-to-stop trip interval i are calibrated independently. Suppose that after the k-th experiment, we need to find parameter configuration for $\Theta_{k+1} = (\theta_{k+1,1}, \theta_{k+1,2}, \cdots, \theta_{k+1,m})$ where $\theta_{k+1,i} = (a_{k+1,i}, b_{k+1,i}), \forall i \in [1, m]$. The direction of the adjustments of a_i or b_i is opposite to the direction of the differences of

[17] Again, value "1" indicates a higher value of l_{Yi} or s_{Yi} (of simulation data) compared to l_{Ri} or s_{Ri} (of EBS data), value "−1" indicates a lower l_{Yi} or s_{Yi}, and "0" being equal.

[18] "1" being larger values (i.e., move upwards or scale up), "−1" being smaller values (i.e., move downwards or scale down), and "0" being no adjustment.

comparing Y_i and R_i only, i.e., $f_d(d_i) = -d_i$ where d_i is d_{li} or d_{si} (see Eq. 6.6). The new values of a_i or b_i for the next experiment are

$$\begin{aligned} a_{k+1,i} &= a_{k,i} + \Delta a \, f_d(d_{li}) \\ b_{k+1,i} &= b_{k,i} + \Delta b \, f_d(d_{si}) \end{aligned} \tag{6.9}$$

where Δa and Δb are specified calibration steps for a and b. The values used in the test case are $\Delta a = 0.05$ and $\Delta b = 0.1$. The start values of a_{1i} or b_{1i} are $a_{1i} = 0$ and $b_{1i} = 1$ (for the first experiment in the second stage).

6.4 A Calibration Test Case

This section presents a calibration test case using the two stage procedure discussed in § 6.3.2. We choose a small model of the HTM service in The Hague composed of ten stops and eight stop-to-stop trip intervals in two directions. The stop names and IDs are listed below. Two sources are placed before *Malieveld* (2810) and *Riouwstraat* (505) respectively which generate vehicle models according to the corresponding timetables. Recall that, in a source, a vehicle model v_j is assigned with a MSR $\varphi'_j \in \phi' \sim \mathcal{N}(\mu, \sigma^2)$ and this ratio is modified by a_i and b_i at each stop-to-stop interval $i \in [1,8]$ so that $\varphi_{j,i} = a_i + \mu + b_i(\varphi'_j - \mu)$ (§ 6.3.1). After each calibration experiment, the distributions of the eight interval driving speeds Y_i from the model are compared with those of the corresponding R_i from the EBS database.

Riouwstraat		*Laan Copes v Cattenburch*		*Javabrug*		*Dr Kuyperstraat*		*Malieveld*
506	$\xleftarrow{4}$	513	$\xleftarrow{3}$	519	$\xleftarrow{2}$	2309	$\xleftarrow{1}$	2810
505	$\xrightarrow{5}$	514	$\xrightarrow{6}$	520	$\xrightarrow{7}$	2310	$\xrightarrow{8}$	2811

6.4.1 Measures of Scale Parameter Differences

In § 6.3.2.3, we mentioned that standard deviation σ and kernel bandwidth h both can be measures of scale parameter differences between Y_i and R_i. To assess their difference, we ran two separate AMCPs with σ and h respectively while the other (model and calibration) settings are identical. The simulation time[19] is 7:00~9:00 a.m., which is however not particularly important in this case as we intend to compare the effect of σ and h. Two weeks (week days only) driving speed EBS data of September 2011 that corresponds to the simulation time and stop-to-stop intervals are queried for model calibration.

[19] The simulation time refers to the effective simulation time which excludes the warmup. Since the vehicle models are generated at the sources, they need some time to disperse over the network. The warmup length depends on the size of the network. Small models with around 5~10 successive stops needs around 10~20 minutes. Larger models comparable to the size of the The Hague network model take around 1 hour. Because we use an output database to log the state updates in the simulation, we can simply adjust (the start time in) the queries to obtain the desired datasets for statistics.

For the test case, no warmup time is required since the model contains no crossings.

The maximum number of experiments in an AMCP is set to be $\bar{k} = 10$ (§ 6.3.2.2). It turned out that each AMCP indeed ran 10 experiments (cf., § 6.4.2). We therefore obtained 80 p-values from 80 KS tests of Y_i and R_i for each AMCP. The differences of both 80 pairs of the datasets are plotted as shown in Figure 6.3; the difference measures are (in order of the legend labels) bandwidth, standard deviation, minimum, 25

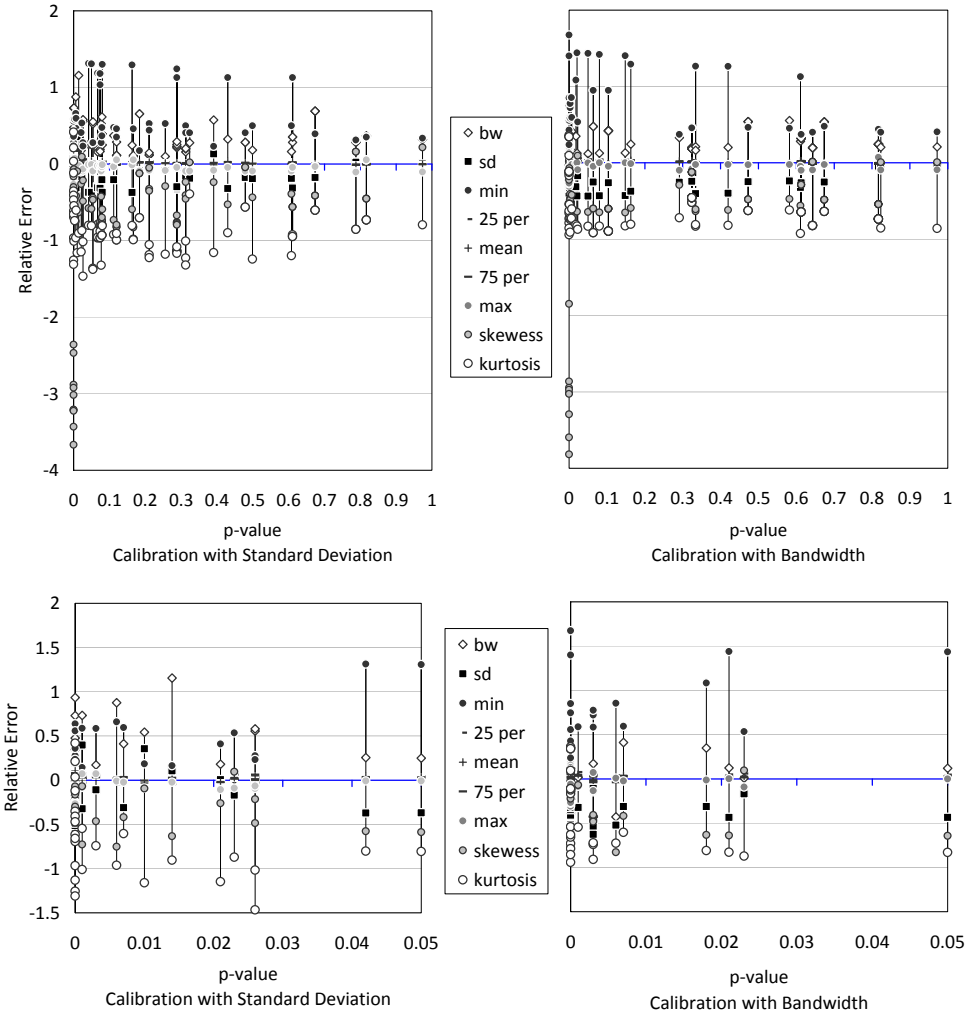


Figure 6.3: Relative errors between the model output and the EBS data in the calibration with standard deviation σ or kernel bandwidth h as a measure of scale parameter differences

percentile, mean, 75 percentile, maximum, skewness and kurtosis. A vertical line connects the measures from the same experiment. It is aligned with the p-value from that experiment. The first two plots have $p \in [0,1]$, and the last two zoom into $p \in [0,0.05]$ since 0.05 is the significance level. Note that we did not use absolute values, i.e., Error = (Model − EBS)/EBS, since the "direction" of the differences are of interest.

Similarities and Differences of Outputs In general, using σ or h as a measure of scale parameter differences gives similar results in the test. In both cases, the model output data has good central tendency. The mean, 25 percentile and 75 percentile are all piled up (and somehow hidden) around the zero x-axis. This is understandable since the μ difference is used to adjust the location parameter. The differences of maximum values are also minor which can be explained by the physical constrains of vehicle driving. The model, however, systematically has higher minimum values which make the largest difference among all the measures (see the dark dots at the top of the plots) with a few exceptions. This means that the EBS data[20] has longer left tails. Since the model does not simulate accident or exceptional situations, this outcome is not surprising either. The EBS data indeed often has negative skews. As the model output data is less negative skewed, the relative error of the skewness is negative too.

There are some differences among the two calibration results too. The relative error of the minimum values in the calibration with standard deviation σ is slightly lower than that in the calibration with bandwidth h, while the relative error of the kurtosis in the former is slightly higher than that in the latter. This means that using σ is more in favor of the tail values and using h is more in favor of the peakedness. This result is consistent with the characteristics of σ and h. A large σ results from large variances. If some data has large variances and long tails, the density curve would be more peaked, which means a relative small h, compared to the data that has the same variances but shorter tails. In such cases, it would make difference whether to use σ or h as a measure of scale parameter differences.

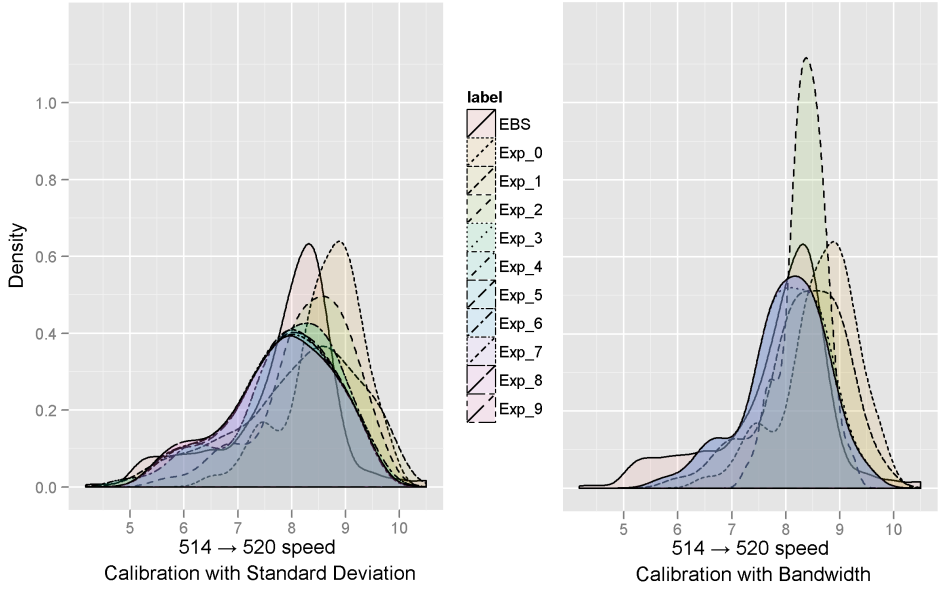
More Discussion on Differences We can illustrate this analysis from the calibrated parameter values and their outcomes in the test case. Table 6.1 lists the calibrated (μ,σ) and (a_i, b_i) and the resulting p-values from KS tests, i.e., from the last (9-th) experiment of each AMCP. Note that (μ',σ') are held in source models, and stop-to-stop intervals hold (a_i, b_i) values. The (shaded) modified (μ,σ) values are listed for the convenience of comparison. They are calculated with $\mu_j = \mu' + a_j$ and $\sigma_j = b_j\sigma'$.

Comparing the two (shaded) columns of modified (μ,σ), there are differences between the results of the two calibrations. For example, the 6-th interval, $514 \rightarrow 520$, has $(0.70, 0.12)$ calibrated with σ and $(0.70, 0.09)$ calibrated with h, where σ_6 in the former is greater. The p-value of the former is 0.026 while the latter has 0.613. The latter is better evaluated by the KS test. Recall that $\alpha_p = 0.05$. (This however does not mean that the latter result is better in a general sense. The KS test is known to be more sensitive near the center of the PDF than at the tails.)

[20] Note that we queried four weeks data.

	MSR	Calibration with Standard Deviation (μ,σ)	(a_i,b_i)	p	Calibration with Bandwidth (μ,σ)	(a_i,b_i)	p
	(μ',σ') **at source**	(0.8, 0.10)	–	–	(0.75, 0.10)	–	–
Modified (μ,σ)	1 2810→2309	(0.75, 0.11)	(-0.05, 1.10)	0.120	(0.75, 0.10)	(0.00, 1.00)	0.817
	2 2309→519	(0.75, 0.11)	(-0.05, 1.10)	0.079	(0.75, 0.10)	(0.00, 1.00)	0.473
	3 519→513	(0.75, 0.11)	(-0.05, 1.10)	0.074	(0.75, 0.09)	(0.00, 0.90)	0.334
	4 513→506	(1.20, 0.18)	(0.40, 1.80)	0.000	(1.10, 0.14)	(0.35, 1.40)	0.000
	5 505→514	(0.80, 0.11)	(0.00, 1.10)	0.211	(0.75, 0.10)	(0.00, 1.00)	0.643
	6 514→520	(0.70, 0.12)	(-0.10, 1.20)	0.026	(0.70, 0.09)	(-0.05, 0.90)	0.613
	7 520→2310	(0.75, 0.11)	(-0.05, 1.10)	0.313	(0.75, 0.10)	(0.00, 1.00)	0.324
	8 2310→2811	(0.80, 0.10)	(0.00, 1.00)	0.288	(0.75, 0.10)	(0.00, 1.00)	0.104

Table 6.1: Calibrated (μ,σ) and (a_i,b_i) and the p-values using standard deviation σ or kernel bandwidth h as a measure of scale parameter differences



p=0.026 p=0.613

Figure 6.4: Density functions of the speed (m/s) of stop interval 514 → 520 compared with that of the corresponding EBS data: model calibration with standard deviation σ or kernel bandwidth h as a measure of scale parameter differences

Figure 6.4 shows the PDFs of the two model outputs (of interval speed) compared with that of the EBS data. The PDF of the EBS data has its density peak at a bit over 0.6 (solid outlined). The model output confirms that the speed is positively correlated with ϕ. Larger σ value of ϕ resulted in larger σ value of Y. The calibration with σ pulls the model output density more towards the tail with a peak at around 0.4, while the calibration with h pulls the density towards the peak at around 0.55. Comparing the other output interval speed pairs shows similar results for EBS data that has large variances and long tails.

Remark The choice of using standard deviation σ or kernel bandwidth h as a measure of scale parameter differences should be made according to the goal of the simulation study. σ can be good when the tail values shall be put into consideration for goodness-of-fit. Examples are risk analysis and assessment, studies that shall include the "uncommon" situations. In such cases, a test statistic that is sensitive to the tail values shall be used instead of the KS test, e.g., a modified KS test (MASON and SCHUENEMEYER 1983) or Anderson-Darling test (ANDERSON and DARLING 1952).

In the LIBROS simulation model, there are no "build-in" disturbances although they can be introduced at a later stage as a model component. The vehicle models in the test case are driving under undisturbed situations. In this case, it is reasonable not to consider the tail values for the calibration of MSR. In general, using h can be good when analysts choose not to include the tail values for analysis but concentrate on the values which represent the "common" situations. In the design and analysis of timetables, e.g., percentile driving time values are often chosen as a reference (TAHMASSSEBY 2009, VAN OORT 2011) where the disturbed situations are excluded.

6.4.2 Bounds of Parameter Configuration

Calibration tries to align the model outputs with the measurements of real systems in order to obtain good parameter configurations. This is a two-way street which requires analysts to respect the model and to respect the measurements at the same time (COOLEY 1997). On the one hand, the parameter configuration should be set such that the model behavior matches the features of the measured data in as many dimensions as there are unknown parameters (ibid.). On the other hand, the configuration should be restricted with bounds that are consistent with the existing theory and the domain knowledge (LAW 2007). Essentially, the goal of calibration is to find good parameters; trying to match outputs with measurements is a means.

Poor Match of Datasets In the previous comparison (§ 6.4.1), there is an example of poor match of model output and system measurement. It is at the 4-th interval $513 \rightarrow 506$ where $p = 0$ (Table 6.1). We will only discuss the case calibrated with h. (The σ case is the same.) At the 4-th interval, μ and σ have large location shift and scaling ($a_4 = 0.35, b_4 = 1.40$), but the output KS test match is very poor. This indeed can be

	h	skew	kurt	μ	σ	min	25%	50%	75%	max
Y_4	0.089	-1.871	3.737	8.591	0.388	7.061	8.534	8.710	8.835	8.964
R_4	0.196	0.620	2.787	8.843	0.840	5.707	8.348	8.885	9.280	12.217

Table 6.2: Data descriptions of Y_4 (model, 10-th experiment) and R_4 (EBS)

confirmed by the data descriptions of Y_4 (model, 10-th experiment) and R_4 (EBS) shown in Table 6.2.

As stated in § 6.3.1, ϕ is defined as maximum speed ratio where $m = \phi\bar{m}$. The "ideal" value of $\varphi \in \phi$ is 1, which means that drivers drive strictly according to speed limit regulations. The randomness in ϕ is represented by $\mathcal{N}(\mu, \sigma^2)$ which has initial values $(0.8, 0.1^2)$ and then adjusted by (a_i, b_i) for location and scale. A question is then what is the (upper) bound of ϕ. We got the information from experts that in free lane driving areas (where the maximum allowable speed $\bar{m}$ is 50 km/h; see § C.1) the actual speed may get to as high as $60 \sim 70$ km/h. Therefore, for experimental purposes, we used $[0.5, 1.5]$ as the bounds of ϕ in the test case such that if an modified $\varphi \in \phi$ exceeds $\varphi_{max} = 1.5$, it is set to φ_{max}, i.e., the exceeded value is cut-off.

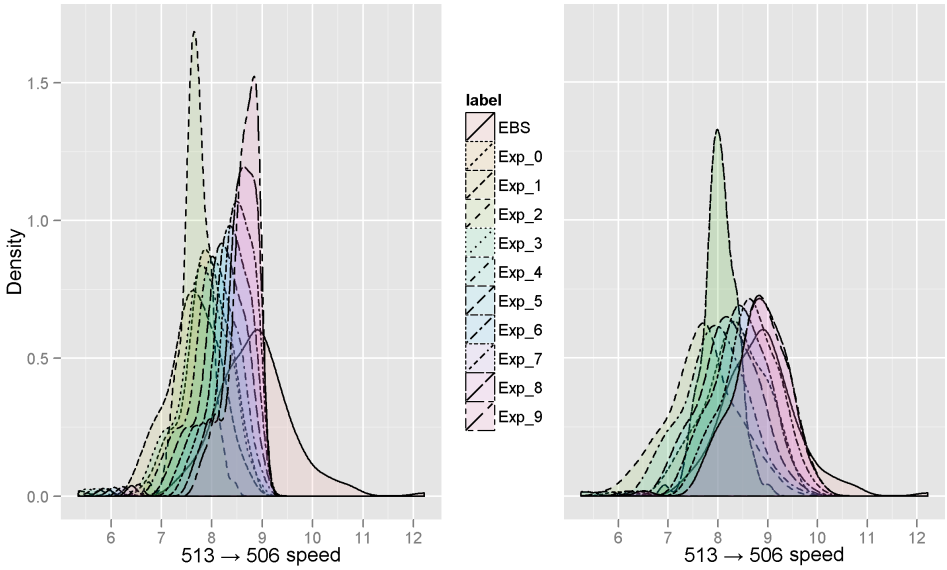
After running the AMCP, the result (Table 6.1, **Calibration with Bandwidth**) shows that most values of $\varphi \in \phi$ do not exceed $\mu + 3\sigma < 1.1$, except for those in the 4-th interval which has a large right shift of location and up-scaling, and poor match.

The PDFs of Y_4 through the experiments ($\varphi \in [0.5, 1.5]$) are shown by the left plot in Figure 6.5. The situation is that R_4 has a location which the simulation model can not produce with the specified bound $\varphi_{max} = 1.5$. Since μ_{Y4} is systemically smaller than μ_{R4}, the calibration tries to push Y_4 upwards by a larger a_4 in each experiment (in stage two). But since the adjusted φ values that exceed φ_{max} are cut-off, σ_{Y4} gets smaller. So the calibration also tries to scale up Y_4 by a larger b_4 in each experiment, obviously without any success.

Discussion Is the upper bound of ϕ too low? The answer is positive if we only look at the two datasets and try to match them. With the bound $\varphi_{max} = 1.5$, the model can produce Y_4 whose highest value is just over the average of R_4 (left plot, Figure 6.5). This means that φ_{max} has to be significantly larger if we want to match the two datasets. Indeed, when we set $\varphi_{max} = 2.0$ and ran the AMCP, the result has a better match (right plot, Figure 6.5). But this loosened upper bound means a doubled speed limit, which is questionable when we intuitively consider the reality.

Is the model wrong in some way? We calculated the speed analytically with the upper bound and the interval distance (which is ~230 meters). The calculation agrees with the model output. Then we checked the mapping consistency, e.g., whether the output and EBS database queries use the corresponding driving time and stop interval, and whether the two stop distances are the same. No sign of error was found.

This makes us wonder the correctness of the R_4 dataset. We queried more R_4 datasets for other times, e.g., evening-peak, off-peak, weekend. Their PDF locations are so



$\varphi \in [0.5, 1.5]$ $\varphi \in [0.5, 2.0]$

Figure 6.5: Density functions of the speed (m/s) of stop interval 513 → 506 compared with that of the corresponding EBS data: calibration with bound $\varphi \in [0.5, 1.5]$ or $\varphi \in [0.5, 2.0]$

high that they cannot be matched by the model if $\varphi_{max} = 1.5$. The high value is counter intuitive, however, because when the distance is only 230 m, vehicles cannot drive too fast if they halt at the stops.

We turned to the experts on this issue. The driving time of this interval according to the (non EBS) information they have is ∼30 seconds in average which makes an average speed ∼7.7 m/s. This value is close to the model output (see the first density curve in the left plot in Figure 6.5). As for the high values of R_4, we may explain it as following. When a vehicle does not halt at a stop, the detection of passing the stop will be only after 60 meters. This is also true for other stops, but since the distance of the 4-th interval is very short (230 m), it makes a significant difference in this case whether the speed is calculated with the time driven over 230 or 170 m. If we do a reversed calculation with the two distances, the result supports our explanation, e.g., 11 m/s in EBS would become 8.13 m/s which is a reasonable maximum speed.

Remark The bounds of parameter configuration constrain the search space. They should be guided by and consistent with relevant theory and domain expertise. A loosened bound may potentially improve the goodness-of-fit of model output data to system observations. But analysts should be aware of the validity of the bound itself. This

in some way also shows a potential danger of using free parameters, a subject often discussed in literature (COOLEY 1997, SCHUNN and WALLACH 2005), since some free parameters do not have known bound.

6.4.3 Validation of Calibration Results

After model calibration, modelers must ask whether the calibration produces a valid model for the system or whether the model is only representative of the particular dataset used for calibration (LAW 2007). One commonly used approach to validate the calibration results is to use two independent datasets, one for calibration, i.e. the calibration dataset, and the other one for validation, i.e. the valiation dataset (ibid., BANKS et al. 2010). We used this approach for the validation of the test case.

6.4.3.1 Experimental Setup

In this experiment, we again simulate the morning-peak 7:00~9:00 a.m. The driving speeds of the aforementioned stop-to-stop intervals from the model are compared with those from the EBS data. The calibration dataset and the validation dataset are from the

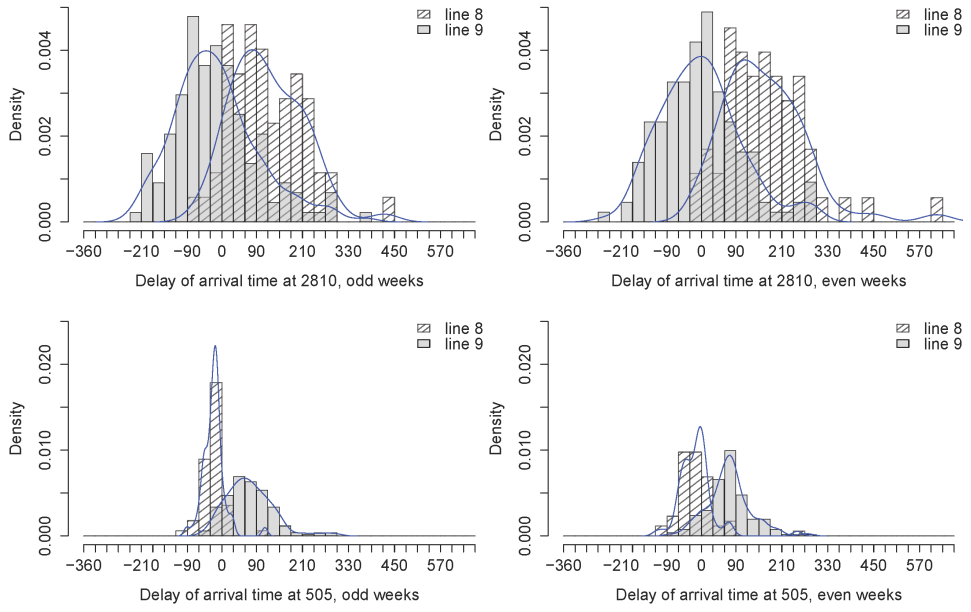


Figure 6.6: Density functions of the delays of the vehicle arrival times (in second) at the stops *Maliveld* (2810) and *Riouwstraat* (505) in the odd and even weeks of October 2011

two odd and the two even weeks (week days only) of October 2011 respectively. § D.1.1 shows the density function plots of the EBS data (the odd vs. even weeks per stop-to-stop interval).

In both cases, the delays of the vehicle arrival times ($t_{delay} = t_{actual} - t_{scheduled}$) at the stops *Maliveld* (2810) and *Riouwstraat* (505) of the corresponding time period in the EBS are used to fit the PDFs of the delays of vehicle generation times in the two source models. Figure 6.6 shows the plots of the density functions. Since there are two service lines (8 and 9) operating between these stops, their PDFs are fitted separately. The stop 505 has shorter distances after the terminals than the stop 2810. Therefore the vehicles arrived in the direction of the former are more punctual than those of the latter (the lower two vs. the upper two plots). The PDFs are used to generate vehicle delay times (with reference to the scheduled times) of the two service lines at the source models for calibration and for validation respectively (the left two vs. the right two plots).

For the calibration, the kernel bandwidth is used as the measure of scale parameter differences (§ 6.4.1). The parameter settings for the calibration experiments are: $\Delta\mu = 0.025$, $\Delta\sigma = 0.025$ with initial values $\mu_1 = 0.8$ and $\sigma_1 = 0.1$; $\Delta a = 0.025$ and $\Delta b = 0.1$ with initial values $a_{1i} = 0$ and $b_{1i} = 1$ (§ 6.3.2.3); the bound of φ is $[0.5, 1.5]$ (§ 6.4.2). The KS test is used as the goodness-of-fit measure in the calibration and the validation (§ 6.1.2 and § 6.3.2.1).

6.4.3.2 Experiment Results

Table 6.3 shows the results of the experiments. (The density plots of the calibration experiments can be found in § D.1.2). For the reasons explained in § 6.4.2, we shall not consider the results of the 4-th interval 513 → 506. The p-values listed under calibra-

		MSR	Calibration (μ,σ)	Calibration (a_i, b_i)	Calibration p	Validation p
		(μ',σ') at source	(0.750, 0.10)	–	–	–
Modified (μ,σ)	1	2810→2309	(0.725, 0.10)	(-0.025, 1.00)	0.482	0.908
	2	2309→519	(0.725, 0.10)	(-0.025, 1.00)	0.090	0.067
	3	519→513	(0.750, 0.10)	(0.000, 1.00)	0.524	0.130
	4	513→506	(0.925, 0.10)	(0.175, 1.00)	0.000	0.000
	5	505→514	(0.675, 0.08)	(-0.075, 1.00)	0.326	0.269
	6	514→520	(0.650, 0.09)	(-0.100, 0.90)	0.279	0.098
	7	520→2310	(0.675, 0.10)	(-0.075, 1.00)	0.747	0.915
	8	2310→2811	(0.750, 0.10)	(0.000, 1.00)	0.054	0.479

Table 6.3: Calibrated (μ,σ) and (a_i, b_i) and the p-values using EBS data from the odd weeks; validated with EBS data from the even weeks in October 2011

tion are from the final calibration experiment. After the calibration, the resulting (μ', σ') in the source models and (a_i, b_i) of the stop-to-stop intervals are used to set the model parameters in the experiment for validation. The p-values of the KS tests from the validation experiment support the null hypothesis that the interval speeds simulated by the calibrated model for the test case have similar density functions (except for the 4-th interval) as the corresponding EBS data.

7

Expert Validation and Evaluation

DURING THE RESEARCH and model development process, we closely followed guidelines of ways to build valid and credible models as discussed in M&S literature (e.g., LAW 2007, BANKS et al. 2010). On the one hand, we researched established theories, e.g., in the domain of rail (and public) transport, graph theory (and graph transformation in particular), and simulation modeling. On the other hand, we actively interacted with subject-matter experts from HTM, take advantage of their knowledge, and obtained first-hand high-quality information on the system structure and system behavior of modeling interest.

The author was present at HTM at a weekly basis, regularly met and interviewed experts from different departments, conducted formal and informal meetings and structured walk-through, where various issues concerning the system structure and behavior and the corresponding model design had been discussed and revised. In this chapter, we will not present the intermediate steps but the final results of the model validation and evaluation of this research.

Since the research objective is to provide a method that automatically generates simulation models with flexible structures using existing data, the evaluation needs to address two major issues:

1. Can the AMG method generate simulation models using data that represent different system structures?

2. Are the generated models valid representations of the corresponding systems?

The first issue can be examined with the developed model generator (§ 5) provided with different infrastructure data that complies with the representational conventions discussed in § 5.2.2. As the main objective in this research, the AMG method is thought of from the onset and is divided into research questions addressed by the fives constructs we proposed (§ 3.1), each of which being presented through the chapters.

The second issue, concerning the structural validity (§ 2.1.4) of the generated models, is also a way to evaluate the AMG method. We choose to use expert validation, since domain experts have good knowledge of the system structure and behavior, and can make good judgment on whether a model resembles an original system (BALCI 1998, LAW 2007) .

7.1 Model Validation Procedure

Expert Panel For the purpose of model validation, the project management of HTM chose a panel consists of nine domain experts. They are managers and senior staffs who have extensive experience and solid knowledge of the HTM transport system including policy, planning, process, product and operations (see § D.2).

Following the advices of the experts, five representative locations in the The Hague light-rail network are chosen for model validation. Each location is supplied with the corresponding infrastructure and service data as discussed in § 5.2 and § 5.3. We generated simulation models with these data and conducted a (final stage) model validation session with the expert panel.

Model Description The five locations are chosen because of their representative or complex infrastructure layouts and busyness in operations:

(1) Wouwermanstraat
(2) Station Hollandspoor (Station HS)
(3) Rijswijkseplein
(4) Bierkade
(5) Central Station (Station CS)

Their infrastructure layouts are shown as model animation snapshots in § C.3. Locations (1) and (4) are examples of "Y" and "T" crossings (§ 5.2.3.2) respectively, which are two common urban rail infrastructure composites. Misc-crossings, turnouts and diamond crossings (§ 5.2.4.1) are used in the model generation of the other three locations. Each crossing model has a control unit (§ 4.3.4.5) that coordinates signals and points in that crossing (§ 5.3.1.3). The vehicle models (generated during simulation) are as specified in § 4.3.3. The vehicle to vehicle and vehicle to infrastructure communication mechanism is as specified in § 4.3.1.

Model Animation and Legend Animation is an often used means of model presentation in expert validation (GONZALEZ 2010). Since the animation in LIBROS has good rendering of model state transitions and interactions (§ 4.5), we chose to record the model animations of the above mentioned five locations and present the videos to the expert panel. An animation legend (§ B.3) is prepared to explain how certain animation features are related to the model states.

Validation Questionnaire A questionnaire (§ D.3) is designed in connection with the animation videos. It is composed of five sections each of which contains questions about the model behaviors in the corresponding simulated locations. The questions concern:

- vehicle models' acceleration and deceleration behavior and stopping position,
- vehicle models' driving and halting behavior (in terms of interaction),
- sensor and point models' behavior, and
- general questions such as whether abnormal model behaviors are observed.

The infrastructure models have a generated CCH (§ 5.1.1). The misc-crossings do not have pre-specified structural patterns (§ 5.2.4.2). The control units (§ 4.3.4.5) in the crossing models have automatically configured check tables (§ 5.3.1.3). The vehicle models use DSDEVS connecting to the infrastructure models (§ 4.3.5.3) which form the backbone for the model communication mechanism (§ 4.3.1). All these elements are relevant to model structures with affect model validity. Only the correct generation of these elements with valid atomic model behaviors as a whole can yield valid model behaviors and interactions as a whole. Therefore, we designed specific questions about the model behaviors and interactions in order to evaluate the structural validity of the models.

Validation Procedure The procedure of the validation session is the following. The animation videos (§ C.3), legend (§ B.3) and questionnaire (§ D.3) were first sent to the individual panelists. They had about two weeks time to complete the questionnaire. Upon the return of the questionnaires, we studied their feedback.

Short after the questionnaire evaluation, a group meeting with the panelists was carried out. We started with a short presentation, explained conceptually the model generation procedure and the structure of the simulation model to the panelists. Some small examples of model communications and crossing controls were also presented. The meeting proceeded with in-depth discussions on the model behaviors and interactions while demonstrating the animation videos, and then with open discussions on the model's structural validity. Although the objective of the session was validation, through the meeting it was also possible to get some feedback from the panelists on the further development and potential use of the simulation model.

7.2 Expert Validation Results

This section reports the results of the session meeting with the panelists. The discussion covers the content of the questionnaire, so we will not report the results of the questionnaire in a separate part.

7.2.1 Driving Behind A Vehicle

Questions 1, 7 and 19 are related to a vehicle's driving behavior in terms of decelerating and stopping behind another vehicle (1) or following closely (7) or cruising (9) behind another vehicle. (In § 4.3.3.3, the principle of vehicle movement computation is explained.) The movements, as can be observed in the videos related to these questions, are in principle correct. But several panelists expressed their concern that in the movement related to Question 7, the second vehicle follows too closely and perfectly to the preceding vehicle. It is clearly caused by the "perfect" numerical calculation of the movement. This closeness does not affect much the driving time but does negatively affect the model credibility. The discussion was then extended to the configuration of vehicles' following-distance in the model in general. We agreed upon a following-distance allowing two seconds reaction time and with a minimum of five meters.

Follow-up We adjusted the configuration of vehicles' following-distance accordingly.

7.2.2 Double Halting at Stops

Vehicle models' halting behaviors can be observed in the animation videos of all five locations, and they are asked by questions 3 (b) and 4. The halting times at the stops are modeled by the corresponding probability distributions whose configurations are defined by HTM experts. Although the panelists all agree that the configurations could be improved using EBS data, they also acknowledge that this refinement shall be attributed to future work, while this research focuses on the AMG of model structure and behavior. Then vehicles' double halting behaviors are discussed. Double halting refers to a vehicle's halting for a second time at a stop which has more than one halting places. Figure 7.1 shows an example at *Wouwermanstraat* (Location 1) where two vehicles (V_1 and V_2) are halting one after another. As we can observe, the stop has two halting places such that the stop is longer than, e.g., the stop on the other side where a single vehicle is halting.

When a vehicle arrives at a double-halting-place stop, it may only be able to halt at the second halting place (which is the case of V_2). Afterward, when the vehicle gains access to first halting place (where V_1 is), it may halt there for a second time for the convenience of passenger boarding. According to the panelists, there are about 15 stops that have more than one halting places in the The Hague light-rail network, but at the time of this validation session there is still no formal regulation of whether the vehicles must, should best or shall not have double halting at these locations. In practice drivers often do so when there are passengers arriving.

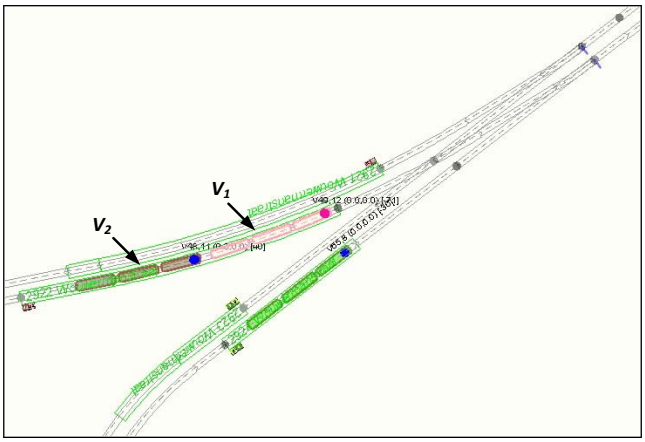


Figure 7.1: Double halting places: an example at Wouwermanstraat

Follow-up We modified the vehicle model such that it has a configurable double halting behavior. The default configuration is to have double halting while this can also be disabled.

7.2.3 Boundary Locations of Crossings

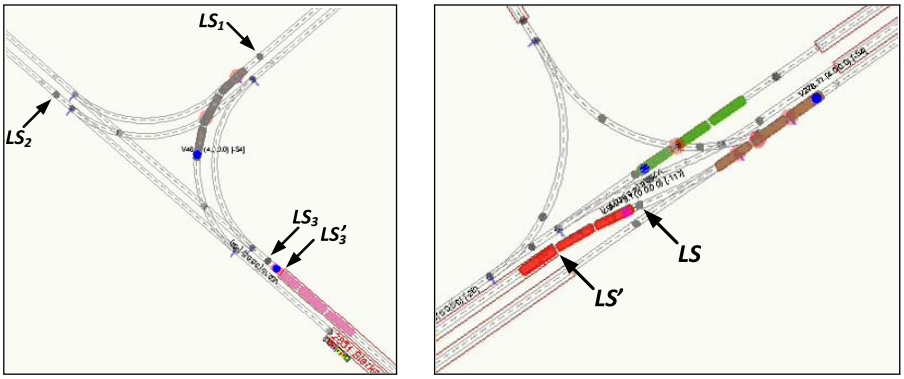
Questions 2, 8, 9, 10 and 18 ask about senor (or point) behaviors and vehicles' waiting positions at the corresponding crossings. The intention of these questions is to check the suitability of the boundary locations of the generated crossings and the effect. In § 5.3.1.2, the principle of model instantiation of a crossing is explained with an example. A lineside signal is added to each entry of a crossing model, i.e., it is right "at" the start node of the track segment that leads to a (facing, trailing or crossing) point (see, e.g., Figure 5.19 D). This means that the "physical position" of the signal depends on the length (and location) of this track segment which is transformed from the corresponding entry node of the hyperedge. Since a vehicle waits in front of a signal, the length of this track segment affects the waiting position of the vehicles.

Does this matter? Sometimes. Figure 7.2 (A) illustrates an example at *Bierkade* (Location 4). It is a "T" crossing with three lineside signals ($LS_1 \sim LS_3$). The location differences of LS_1 and LS_2 (in terms of meters) obviously do not influence much the simulation output. However, the location of LS_3 can make a difference because it is short after a stop (animated as red outlined track).

Why does this matter? When a vehicle waits at LS_3 for another vehicle to pass, the waiting time counts into the "gross" driving time between two stops. But in fact, the vehicle shall wait at the stop (i.e., the location of LS_3 shall be pulled backwards to LS_3'), by which this waiting time would be counted into the halting time instead of the "gross" driving time. This time calculation difference can be significant when the stop is at a

busy location. Figure 7.2 (B) illustrates another example at *Station HS* (Location 2), in which the waiting vehicle (at *LS*) has a large part of its body outside of the stop. For the same reason, the position of *LS* shall be at *LS'*, so do the lineside signals next to the other three stops in this illustration.

Follow-up One way to solve this problem is to modify the model generation such that for a crossing hyperedge that is short behind a stop hyperedge, the corresponding entry (track) vertex of the former shall be replaced with the track vertex that is attached to the exit vertex of the track hyperedge. This can be added into transformation step 2 (§ 5.2.4.1) as a final sub-step (ix).



(A) "T" crossing at Bierkade (B) Misc-crossings at Station HS

Figure 7.2: Boundary locations of crossings

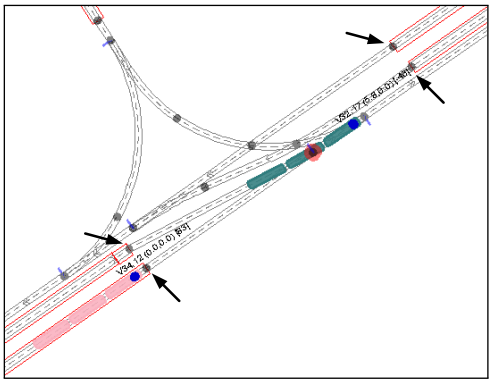


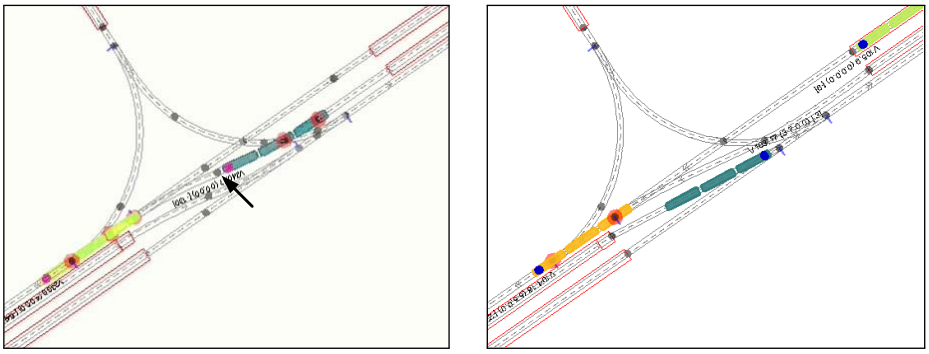
Figure 7.3: Adjusted boundary locations of misc-crossings at Station HS

Another way is to modify the CAD data. We can prolong the corresponding track entity that will become the entry vertex of the crossing, and connect it directly with the stop entity (§ 5.2.2) where we wish to place the signal. This means that any track entity that was in between must be removed. We chose this solution since it is simpler, and modified the CAD data accordingly. Figure 7.3 shows an example of the improved model generation outcome at *Station HS*. The arrows in the figure indicates the repositioned signals (gray dots) which are the adjusted boundary locations of the misc-crossings. Compared to the corresponding signals shown in Figure 7.2 (B), the adjusted positions were "pulled back" to be right at the ending positions of the stops.

7.2.4 Search Distance of Misc-Crossings

Question 8 refers to a situation that is a bit different than those of the other questions mentioned in § 7.2.3. As illustrated in Figure 7.4 (A), at *Station HS* a vehicle is waiting at a crossing (where the signal position is indicated by the arrow) for another vehicle to pass first. As stated in § 7.2.3, the vehicle shall wait at the stop (behind it). But in this case, the essential problem is not the same as those discussed in § 7.2.3.

In Figure 7.4 (A), one can observe that the waiting vehicle is lying on a crossing point (where it would block other vehicles to pass), while it is waiting in front of a signal that guards a trailing point. This means that the crossing point and the trailing point are not merged into the same misc-crossing when they should. Similar problems are found at *Rijswijksplein* (Location 3). We suspect that the value of the search distance in the misc-composite finding (MCF) algorithm (§ 5.2.4.2, bounded path distance $\boldsymbol{b} = 10$) is too short.



(A) Misc-crossings at Station HS
Search distance 10 meters

(B) Adjusted misc-crossings at Station HS
Search distance 20 meters

Figure 7.4: Search distance difference of misc-crossings – an example at Station HS

Follow-up We adjusted the configuration of the bounded path distance ***b*** in the MCF algorithm from 10 to 20. Figure 7.4 (B) shows the improved model generation outcome at *Station HS* as an example. We can see that the gray dots which indicates the signals in the middle of this crossing are no longer present, which means that the points are merged into the misc-crossings with an appropriate aggregation level. Together with the improvement discussed in § 7.2.3, the vehicle models are waiting at the stops for other vehicles to pass the crossing. This is the desired result.

Note that it is also possible to achieve a similar result by modifying the CAD data. We can prolong the track entity that will become the exit vertex of a crossing. As such, this crossing will have a "shorter" path distance to a connected crossing. When this distance is within ***b***, they will be merged into the same misc-crossing. This alternative solution can be useful if certain search paths are longer than ***b***, but we do not want to increase ***b*** to affect the merge result of some other misc-crossings.

Since infrastructure layouts can have many varieties, there is no standard of the "best" value of ***b***. A safe way to assure appropriate misc-crossing merging is to inspect the model generation outcome, and to adjust the ***b*** value or the CAD data according to the particular situations.

7.2.5 Control Logic at Crossings

Questions 3(a), 9(b), 14, 15 and 20 concern the control logic (§ 4.3.4.5) at the crossings. They ask about the corresponding interactions between vehicles, signals and switches in order to check the correctness of the control units (logic) and check tables (§ 5.3.1.3). The discussion was held in conjunction with the issues in § 7.2.3 and § 7.2.4 since they are closely related.

For what can be observed from the animation videos related to the questions, the control logic for the crossings are correct, except for the situation that is related to question 9(b). In this question, two (successive) vehicles from the same direction want to pass a crossing and drive towards the same direction. The second vehicle was waiting for the first vehicle to pass first then enters the crossing. In this case, however, it shall pass the crossing following the first vehicle without waiting. This is caused by the following reason. In processing an access request, the control unit checks whether the points required by the requested route are free. When they are occupied, then the request is not granted until the points are freed again. This is the situation in the example of question 9(b).

Follow-up To solve this problem, we introduced an *active route* variable into the check table to register a granted request if the requesting vehicle has not yet left the route. The control unit shall check this variable and grant access if a requested route is an active route. This change made the relevant model behavior plausible. Note that the control unit and check table presented in § 4.3.4.5 already figure the improved version.

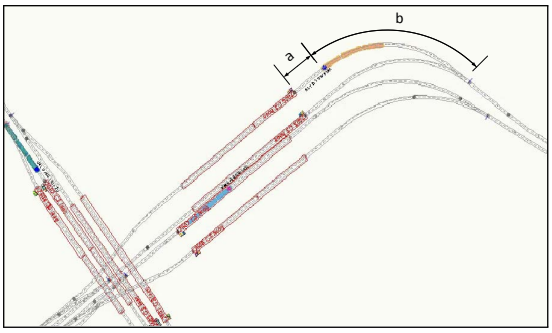


Figure 7.5: Short change of maximum speed – an example at Station CS

7.2.6 Other Remarks

Some panelists pointed out that in the *Station CS* (Location 5) animation video, some vehicles do not drive very smoothly. Figure 7.5 shows the scenario where at the upper right corner there are a number of curved tracks. In the model, the maximum (allowable) speed in the city is in general $\bar{m} = 50$ km/h (13.89 m/s); for the curves, the maximum speed is calculated with $\bar{m} = \sqrt{r \cdot a_{max}}$ m/s where r is the curvature radius in meters, and $a_{max} = 0.65$ m/s^2 is the allowable centrifugal acceleration based on passenger comfort and wearing of the wheel and track[1]. This simply means that the maximum speed in curves is less than on straight tracks. The situation of unsmooth driving in *Station CS* is that, as shown in Figure 7.5, after the curve b, there is a short straight track a, where vehicles accelerate for a short while and then decelerate again because a is followed by a stop. Such drivings in practice would be a little smoother. Some other constructive comments on certain behavior details are also made, which we will not elaborate one by one. Although many improvements in the model can be made, the panelists also acknowledged that this type of behavioral details do not alter much the simulation output.

The panelists also suggested further directions for model development. For example, new model components for modeling tail tracks, disturbances, rerouting, transfer, depot; using EBS information for model halting times, and OV-chipcard[2] information for passenger origin-destination, etc. They foresee a wide range of potential model uses such as driving time analysis, timetable analysis, capacity study, optimization of operations, among many others.

[1] The formula and value of a_{max} are based on HTM internal documents.

[2] The *OV-chipkaart* (in Dutch) is the electronic payment method for public transport in the Netherlands (www.ov-chipkaart.nl).

7.3 Reflection

In the validation session, the panelists showed genuine interests in the model as well as the inner working of the model, and appreciated the fact that complex models as such can be generated using infrastructure data. As follow-ups for the validation session, minor adjustments in the model and parameters adjustments for the model generation are made to fine-tune the AMG outcomes, which to a certain extend shows the flexibility in the model and model generation design. Besides these minor adjustments, there are no substantial modeling problems raised by the experts in the validation session. Overall, the experts support the structural validity of the model for simulation studies such as timetable and driving time analysis.

The validation session showed the structural validity of the LIBROS models generated by the AMG method. Founded on rich systems theory, M&S theory, graph and hypergraph transformation theory, and accompanied by practical applications of the method (and the generated models, § 8.2) in the domain of light-rail transport, we argue that the method has a strong potential for applications to graph-representable systems such as those in the infrastructure domain.

8 Epilogue

In this research, we studied Automated Model Generation (AMG). The modeling of increasingly larger and more complex systems is one of today's challenges in the field of Modeling and Simulation (M&S) (Crosbie 2010). Many examples show that complex models take long to develop and incur high costs (e.g., Wieland and Pritchett 2007, Longo 2011). While there is a rich history of efforts to improve modeling processes, there is still considerable room for improvement (Fowler and Rose 2004). Along with the advances in data collection technology and more popular use of computer-aided systems, more and more data has become available in many organizations (Glotzer et al. 2010). This allows for a higher degree of automation in modeling. At the same time, the increased amount of data often requires a certain degree of automation since the data can no longer be manually handled effectively and efficiently (ibid.).

The research presented in this thesis developed a method that can use existing data to automatically generate complex simulation models. The research objective is *to provide a method that automatically generates simulation models with flexible structures using existing data assuming that these simulation models are intended for a certain domain.* By "generating simulation models with flexible structures", we mean that simulation models are not generated by parameter-based configuration on a pre-specified model structure, but the model has a structure that is dynamically constructed according to the existing data during the AMG.

As discussed in § 1, the AMG method in this thesis differs from other AMG research in at least two aspects. First, the data used for the AMG does not contain specifications

of the model structure to be generated. Second, the generated models have structures that are dynamically constructed. This entails that there is an information gap between what is contained in the data as the AMG input and what is required by the AMG in terms of model structure. The AMG method thus is in need of functionalities that can close this gap. Additionally, to be able to dynamically structure model components in various combinations into valid simulation systems (PETTY and WEISEL 2003a), the AMG method must have composable model components at its disposal. Although the statement of "components should support composition" (CRNKOVIC et al. 2011, SZYPERSKI 2011) may seem redundant, many researchers acknowledge that model composability is difficult to apply (YILMAZ 2004, SZABO and TEO 2007, TOLK et al. 2010). To tackle these challenges in the research, we decomposed the research objective into four interrelated tangible parts and defined the following research questions (RQs):

1. *What is a good way to define flexible structures for simulation models in order to achieve the research objective?*
2. *What are the requirements for the data in order to achieve the research objective?*
3. *What functionalities should a method provide in order to automatically generate simulation models with flexible structures using existing data?*
4. *What is the quality of the simulation model generated by the method?*

The first three questions concern the AMG method itself. Since the method aims at generating domain simulation models from existing data, we have to know first what are the AMG inputs and what has to be generated as an output, and then define the functionalities the AMG method shall deliver. The last question concerns the quality of the simulation model generated by the AMG method, through which the quality of the method is also evaluated.

8.1 Research Findings

In § 3.1, based on the concepts and theories presented in § 2, we proposed five constructs for the AMG method as a first step towards tackling the research problem.

1. *A domain meta-model (or meta-models).*
   It should represent corresponding knowledge levels of the systems of interest.
2. *A domain model component library.*
   It should conform to the domain meta-models (construct 1), implement the concept of hierarchical component-based modeling, and use an appropriate underlying modeling formalism.
3. *A meta-model (or meta-models) of the data.*
   It should represent the existing data of interest.
4. *A model transformation definition.*
   It should be specified on the meta-model of the data (construct 3) and the domain meta-model (construct 1).

5. *A model calibration procedure.*
   After initial generation, simulation models could be calibrated when empirical data of the systems of interest exists.

Rationale To be able to define flexible model structures (RQ1), a domain meta-model and a domain model component library (constructs 1 and 2) are proposed. The former should allow for an abstract representation of model structures of a class of models. The latter should specify model behaviors that are used as building blocks for model composition. Note that, as shown in § 4, the domain meta-model is then also defined as a part of the domain simulation library.

For the AMG method, existing data is supposed to provide concrete information about model structures and parameterization (RQ2). Since the data should be transformed to generate simulation models, the AMG method requires a meta-model of the data (construct 3) based on which the transformations can be specified.

Existing data typically has quality issues (§ 3.3) and does not contain all types of information, particularly in terms of model structure, that are required by the AMG. This information gap should be first identified and then corresponding measures may be determined to close this gap. The functionalities that should be provided by the AMG method (RQ3) basically include the measures that are automatable. To automate these measures, a model transformation definition (construct 4) should be specified on the meta-models. This is a focus of the thesis.

Model calibration is often a necessary step in modeling. When operational data from the real system is available, calibration could be automated as well. We therefore complement the AMG Method with a model calibration procedure (construct 5). The subject of model calibration, however, is not a focus of the thesis.

To study the AMG method, we used cases in the domain of light-rail transport.

Research Question 1 A good way to define flexible model structures is studied with the first two constructs for the light-rail domain. We designed and developed a domain model component library, LIBROS (Library for Rail Operations Simulation), § 4, that represents different knowledge levels of the systems of interest, implements the concept of hierarchical component-based modeling, and uses an appropriate underlying modeling formalism, where the DEVS (Discrete Event System Specification) formalism is chosen for reasons explained in § 2.4.1.3. In LIBROS, the atomic models specify model behaviors at the elementary level. The coupled models are defined to represent domain meta-models that allow for a set of model compositions.

To achieve a meaningful systems decomposition, we researched relevant literature and discussed and revised the design in an iterative manner with domain experts (§ 4.2). A number of modeling choices and alternatives were evaluated with criteria such as modularity, extensibility, simplicity and computational cost. To support a good level of modularity and composability (§ 4.1), we designed cohesive model ports and coupling relations and a communication mechanism (§ 4.2 and § 4.3). Based on message

propagation, the communication mechanism is suitable for decentralized communications in a connected network model (§ 4.2.3 and § 4.3.1).

A number of other modeling practices are also discussed. A SAM (shared alterable mixin) design pattern (§ 4.3.2.2) is presented for atomic model behavior and function composition, reuse and extension. The pattern is used at three places in LIBROS design, for the sensor behavior (§ 4.3.4.4) and for the operations on constructing model ports and setting up couplings (§ 5.3.1.4). In § 4.3.3.1, we discussed the choice between Quantized DEVS (QDEVS) and Generalized DEVS (GDEVS), two approaches for modeling continuous and hybrid systems based on the DEVS formalism. We then explained the use of GDEVS and the handling of interrupts in GDEVS with examples. Consistent with the DEVS formalism but more constrained in the way an atomic model handles state changes at the arrival of an external event, § 4.3.3.5 presented a state update and transition mechanism (SUTM). The mechanism differentiates the concepts of state update with state transition, and handles them separately with external and internal transition functions. Its benefits and potential computational costs are discussed.

A good design of flexible model structures is the outcome of joint considerations of many modeling aspects: the representation of the knowledge levels of the systems of interest (§ 2.1), the choice of modeling style and formalism (§ 2.4), and the design choices of model structure, behavior and interaction. While the concepts of domain meta-models and model components and the corresponding requirements could serve as a general guidance, a number of practices in the design of LIBROS and the rationale behind the design choices as well as their benefits and computational costs are discussed, which could be useful for modular and composable model design in many other domains. To conclude, a good design of flexible model structures should have:

(1) a generic modeling formalism,
(2) definitions of model structures through meta-models,
(3) separation of concerns of model behaviors through model components,
(4) reusable definitions of model behaviors, and
(5) cohesive definition of model interfaces and interactions.

The following are not necessary but can be beneficial for the design of flexible model structures or the design of simulation models in general:

(1) a modeling formalism that supports modeling discrete and continuous systems,
(2) hierarchical (component-based) modeling,
(3) separation of concerns of model state transition definitions of elementary model behaviors, and
(4) efficient model communications.

8

Research Question 2 As stated previously and in § 3.1, existing data is supposed to provide concrete information about model structures and parameterization for the AMG method. Typically, existing data does not contain all types of information required by

the AMG, particularly in terms of model structure. We need to identify the information gap, and ways (and relevant domain knowledge) to close this gap.

In § 3.3, we discussed data quality categories and criteria based on literature that could help to detect data quality issues. We exemplified the issues in the data used for the AMG of the light-rail models, and measures to solve the issues. A concrete example of the requirements for the data is presented in § 5.2.1.

In principle, data issues related to (semantic and pragmatic) completeness for the AMG are hard to solve automatically. For example, in the AMG of LIBROS models, minor modifications, such as the locations where vehicle models shall be generated in the simulation, are made manually to the infrastructure data to solve the issue of pragmatic completeness. To solve data issues related to (syntactic and mapping) consistency, measures such as data type or format conversion or mapping tables should be prepared in advance to the AMG. These measures can be integrated into the AMG method. To solve semantic accuracy and presentation suitability[1], when relevant domain knowledge and reasoning for deduction are available, they should be formalized and defined as transformation rules in the AMG method. This is a major challenge in the method design.

To conclude, with regard to the information of model structure and parameterization, the data that is provided to the AMG as input should be assessed according to how well the data provides information for model component identification, composition and configuration. (The data quality criteria are discussed in detail in § 3.3.) The requirements for the data (assuming that the data has syntactic accuracy and timeliness) should have:

(1) semantic and pragmatic completeness, and

(2) syntactic and mapping consistency, or conversion rules or mapping tables or something alike that can solve the inconsistency in the data.

Research Question 3 In designing the AMG method, the information gap between the data and the model, particularly in terms of model structure, should be carefully identified, and corresponding measures are then defined to close this gap. The functionalities that should be provided by the AMG method basically include those measures that are automatable.

In generating LIBROS simulation models, infrastructure data is used as a basis for model structures (§ 5.2.1). The major challenge for the AMG method is the presentation suitability of model structures (including identification of model components and compositional relations). To solve this problem, the types of information that are available in the data and those that can be inferred from the data (which are required for the AMG) are arranged by their dependencies (§ 5.2.1.2). Based on these dependencies, three transformation steps (with sub-steps) are designed to infer the required information from the data and to finally generate the simulation models.

[1] Presentation suitability refers to the degree to which the data is appropriate for the purpose of data use in terms of format, unit, precision and type-sufficiency, where type-sufficiency refers to the degree to which the data includes all the types of useful information (§ 3.3.2.2).

The data structure representations (i.e., meta-models of the data) and transformations are defined based on graph transformation theories (§ 5.1). A pair of an ordinary digraph and a hypergraph is used to represent the Model Composite Graph (MCG) which has a Compositional Containment Hierarchy (CCH) of the simulation model to be generated (§ 5.1.1). A CCH is a strictly nested component inclusion hierarchy. A set of transformation rules are specified on the meta-models of the data (including for the intermediate steps) and the meta-model of the simulation model as a final outcome (§ 5.1.2). The meta-models of the data and the data structure compositions of interest (for transformation) are defined as graph patterns and pattern composites whose matching instances are recorded as hyperedges in the MCG (§ 5.2).

In the first transformation step, a digraph that represents the rail infrastructure network (as a flat graph) is constructed from the original infrastructure data (which does not contain vertex relations) based on geometrical inferences (§ 5.2.1 and § 5.2.2). In the second transformation step, graph pattern matching and hyperedge replacement are performed incrementally (with partially ordered sub-steps where composition and decomposition are both used) on the graphs which produce an MCG (§ 5.2.3 and § 5.2.4). For the transformation, we defined a number of graph patterns and pattern composites, an algorithm for ordered composite isomorphism for the matching of the defined graph patterns, and an algorithm for the merging of composites within a defined distance measure. The purpose of the MCG is to aggregate the vertices in the digraph into a CCH such that the matched graph patterns can be transformed into corresponding model components in the next step. In the third transformation step, the vertices and hyperedges in the MCG are transformed into atomic and coupled model components corresponding to the vertex and hyperedge types, and the vertex relations are transformed into model coupling relations (§ 5.3.1 and § 5.3.2). In this context, a hyperedge (composite) is injective homomorphic to the (generated) coupled infrastructure model, where a number of sub-components are added into the model according to the meta-model definitions. Among these sub-components, the control logic of each crossing model component is generated according to the corresponding infrastructure layout. After these steps, we obtain a simulation model where the model behavior at the elementary level is pre-specified in the atomic components in the domain model component library, and the model structure is dynamically constructed using the coupled components according to the infrastructure data.

In principle, transformation rules for the AMG can be defined when the data that serve as input for AMG has semantic and pragmatic completeness, has definable measures for syntactic and mapping inconsistency (if any), and when modelers have sufficient domain knowledge and deductive reasoning for the definition of transformation rules that can solve data issues related to semantic accuracy and presentation suitability (see **Research Question 2**) with regard to model structure and parameterization. The functionalities an AMG method should provide are transformations defined on meta-models of the original data structure (as the AMG input), of intermediate structures and of the simulation model. Suppose that the original data structure is a non-graph, meaning that the data items do not have specified relations (or they can be deemed as a graph with vertices but without edges); and assume that the meta-model of the simula-

tion model and the model components are specified, e.g., in a domain model component library. Then the transformations need three steps: from a non-graph to a digraph, to an MCG, and finally to a simulation model, where the vertices and hyperedges of the MCG should have corresponding (pre-specified) model components. To conclude, the functionalities that an AMG method should provide are transformation rules: (The data quality criteria in (1) are discussed in detail in § 3.3.)

(1) that can solve data quality issues related to semantic accuracy and presentation suitability in terms of model structure and parameterization,

(2) that are defined on the meta-models of the original data structure, of the indeterminate structures, and of the simulation model,

(3) that can construct a representation of the (to be generated) model structure whose components can be mapped to corresponding pre-specified simulation model components,

(4) that can construct a simulation model according to the representation of the model structure.

The following are not necessary but can be beneficial for the design of the transformation rules:

(1) graph and/or hypergraph-based structure representation and transformation,

(2) recursive definition and incremental search of model composite patterns, and

(3) using both composition and decomposition in the transformation rules.

To complement the AMG method, when operational data from the real system is available, the simulation model can be calibrated by a model calibration procedure using user defined goodness-of-fit measures and parameter search algorithms (§ 6). In the calibration test, a preliminary study was made to fit interval driving times of the generated LIBROS light-rail simulation model of The Hague. A number of issues related to the calibration experiments are also discussed. Note that the goodness-of-fit measures in the calibration procedure can serve as a way to validate the relevant model output data (operational validation through comparison).

Research Question 4 During the research and model development process, we closely followed guidelines discussed in M&S literature in order to build valid and creditable models. Since the model component library is intended for the domain of light-rail transport, as a final step of model validation in this research, and a way to evaluate the AMG method, a validation session (with a combination of questionnaires and discussions) was organized with a panel of nine subject-matter experts in the domain of light-rail transport (§ 7). Specific questions were designed to ask the panelists about the model behaviors and interactions of the generated LIBROS light-rail simulation models of The Hague in order to evaluate the structural validity of the models. As follow-ups for the validation session, minor adjustments in the model and parameters adjustments for the model generation were made to fine-tune the AMG outcomes. The ex-

perts raised no substantial modeling problems in the validation session. Overall, the experts appreciated the fact that complex simulation models can be generated using infrastructure data, and showed support for the structural validity of the model for simulation studies such as timetable and driving time analysis. They also suggested further directions for model development.

The validation session shows the structural validity of the LIBROS models generated by the AMG method. The infrastructure models have a generated CCH (§ 5.1.1) as a result of graph pattern matching and replacement (§ 5.2 and § 5.3). The control units (§ 4.3.4.5) in the crossing models have generated structure-based control logic (§ 5.3.1.3). The vehicle models use DSDEVS connecting to the infrastructure models (§ 4.3.5.3) which form the backbone of the model communication mechanism (§ 4.3.1). All these elements are relevant to model structures which affect model validity. Only the correct generation of these elements with valid atomic model behaviors as a whole, can yield valid model behaviors and interactions as a whole. We argue that the design of the AMG method is sound for the following reasons:

(1) the AMG method is founded on rich systems theory, M&S theory, graph and hypergraph transformation theory;

(2) the application of the AMG method to the domain of light-rail transport generates simulation models that were validated by a panel of domain experts; and

(3) the light-rail transport models generated by the AMG method have been used to support real-life decision making, as is exemplified in § 8.2.

Although the models are intended for the domain of light-rail transport, the modeling concepts and practices as discussed in § 4 could be useful for model design in many other domains. The concepts of graph-based composite definitions and transformations, and the algorithms used in the transformation steps § 5 could be applicable to systems that can be represented with graph-based structures.

8.2 Practical Use of LIBROS Models

In this research, we did not focus on model uses. Hence we did not present practical uses of LIBROS models. To show the practical relevance of this research, we name a number of model uses which helped strategic, tactical and operational decision makings in light-rail transport systems.

In CAI (2011), key performance indicators such as deadhead kilometers and average delays were evaluated using LIBROS models to inform the choice of depot capacity and vehicle schedules. To reduce waiting times at the entrance of the tram tunnel *Grote Markt* in The Hague (where the setup is similar to that discussed in § 4.4), simulation experiments were ran with the models using different timetables and safety control measures. The models were also used to study the impact of, e.g., merging two stations and the new infrastructure construction at *Station HS* in the light-rail network of The Hague.

HTM, the public transport service provider in *Haaglanden* region, is interested in further uses of the model generator and the model component library developed by this research. A preliminary version of a graphical user interface was developed (VAN ANTWERPEN 2011) to help end users changing parameter settings of the AMG tool. At the time of writing this thesis, the organization is also in the process of developing a web application to access the simulation models.

8.3 Future Research

With regard to model calibration, this research had a preliminary calibration study of the generated LIBROS simulation models. Since the organization has abundant operational data from the real system, it is possible to conduct in-depth model calibration studies. This is not only relevant to the LIBROS models but also to the research of calibration methods and techniques in general. In M&S literature, calibration methods and techniques, particularly those for large-scale microscopic models, are not often discussed. Many complex simulation models require a long time to simulate. When they have a large number of unknown parameters, optimization-based calibration method can be very costly in terms of computations. Therefore, alternative methods and techniques can be useful for these models when the computational cost is an issue for model users. Many design issues related to calibration experiments can be studied, e.g., how to separate unknown parameters into calibration stages and what are the pros and cons of making such design choices.

For the LIBROS library, model components and other components such as user interfaces, statistics and reporting tools can be further developed. Current development focused on a basic set of infrastructure model components. New components can be developed, e.g., to model tail tracks, single tracks, block systems, disturbances, rerouting, depot, and to model the origin-destination of passengers. Some of the components can be linked with real data, such as the data from the EBS and OV-chipcard systems.

From the viewpoint of model generation, multi-resolution and multi-perspective model generation can be an interesting and challenging future direction. The development of a domain simulation model or a class of domain simulation models is often intended for similar purposes of study. When simulation models can be generated with different resolutions and/or perspectives from a library or a set of libraries, such libraries will be highly reusable. This of course requires substantial research efforts. Above all, we need libraries that contain sufficient domain knowledge and organize different parts of this knowledge in an appropriate manner to allow users or automated agents to query the knowledge with relevance.

Furthermore, the AMG method designed in this research can be applied for AMG of other systems besides light-rail transport systems. Straightforward applications include those in the infrastructure domain such as heavy-rail and road transport, pipeline and grid systems. A step further could be made to systems that can be represented with graph-based structures such as production and supply chain systems.

Appendices

A Background

A.1 Modeling Relation: Homomorphism

A system can be modeled in many ways depending on the purposes of the simulation studies. Even with the same purpose, different modelers would often model a system differently. In general, the relation between a model and its original system (that is modeled) is homomorphic (SHANNON 1975, ZEIGLER et al. 2000, KLIR 2001).

> Homomorphism (from Greek *homoios morphe*, "similar form"), a special correspondence between the members (elements) of two algebraic systems, such as two groups, two rings, or two fields. Two homomorphic systems have the same basic structure, and, while their elements and operations may appear entirely different, results on one system often apply as well to the other system. Thus, if a new system can be shown to be homomorphic to a known system, certain known features of one can be applied to the other, thereby simplifying the analysis of the new system.[1]

More specifically, what does homomorphism mean in M&S? Following KLIR (2001), a homomorphic relation (homomorphism) between a model and its original system is contingent upon a function from relevant entities of the original system onto the corresponding entities of the modeling system (i.e., the model) under which the relation among the entities[2] is preserved. The *onto* function in this definition implies that the

[1] *Encyclopædia Britannica*. Encyclopædia Britannica Online Academic Edition. 2012.

[2] The entities to which the homomorphic function is applied and the relation that is preserved under the function depend on the type of systems involved (KLIR 2001). The "entities" and "relations" are meant in a broad sense. Entities can be parts, states, inputs, etc. Relations can be couplings, state transitions, etc.

entity mapping is surjective, which means that the model is supposedly to be simpler than the original.

To formally state the relation, let $S_0 = (X, \phi)$ be an original system where X denotes a set of entities and ϕ denotes a set of binary relations on X; and let $S_1 = (Y, \psi)$ be a modeling system where ψ is binary relations on entities Y. S_1 is a model of S_0 if and only if there exists a function $h : X \to Y$ such that[3]

$$\forall x_1, x_2 \in X \wedge \langle x_1, x_2 \rangle \in \phi \implies \langle h(x_1), h(x_2) \rangle \in \psi \tag{A.1}$$

This modeling relation is weak (i.e., regular) homomorphic. An additional condition is required for a strong homomorphism[3]:

$$\forall y_1, y_2 \in Y \wedge \langle y_1, y_2 \rangle \in \psi \implies \exists\, x_1 \in h^{-1}(y_1) \wedge x_2 \in h^{-1}(y_2) : \langle x_1, x_2 \rangle \in \phi \tag{A.2}$$

This means that a strong homomorphism requires that each relation in a modeling system is invertible into the original system.

Figure A.1 exemplifies weak and strong homomorphisms from an original system S_0 to a modeling system S_1 (KLIR 2001). In the example, S_0 is a simple unlabeled state transition system with a set of four states $X = \{a, b, c, d\}$ and a set of state transitions $\phi = \{a \to a, a \to d, b \to a, \cdots\}$. The function h maps X to three states $Y = \{\alpha, \beta, \gamma\}$ where the states b and c are merged into β. Under the function, the original state transitions ϕ are preserved in ψ of S_1. The model S_1 would be weak homomorphic if it contains all the state transitions (let them be ψ_{weak}) shown in the S_1 block. To be strong homomorphic, S_1 could only contain the state transitions without the ones that are in dotted lines, *i.e.*, $\psi_{\text{strong}} = \psi_{\text{weak}} \backslash \{\gamma \to \alpha, \gamma \to \gamma\}$.

Homomorphisms can be composed as function compositions. The relation holds because the composition of homomorphisms is again a homomorphism; see, e.g., CLARK (1984) §60 and WARNER (1990) Theorem 12.4 for proofs. Although this is an algebraic concept, it has analogies in M&S, i.e., a modeling system can be the origin of another modeling system. We may denote this relation as following.

$$S_0 \xrightarrow{h_0} S_1 \xrightarrow{h_1} \cdots S_{n-1} \xrightarrow{h_{n-1}} S_n \;\Rightarrow\; S_0 \xrightarrow{h_0 \circ h_1 \cdots \circ h_{n-1}} S_n \tag{A.3}$$

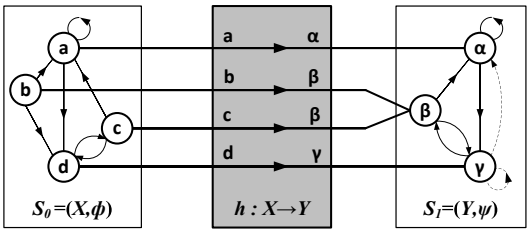


Figure A.1: An example of weak and strong homomorphism (ibid.): the dotted state transitions violate the condition required by a strong homomorphism

[3] Adapted from KLIR (2001) and WALICKI et al. (2001).

A.2 A DEVS Simulator: DSOL and ESDEVS

In § 2.4.1.3, our choice of using DEVS as the underlying formalism for model components is discussed. This section briefly presents the DEVS simulator that is used in this research. The simulator has a DEVS simulation core which is defined on top of a generic event-scheduling based simulation core. Both are developed as Java libraries. The former is called ESDEVS (*Event-Scheduling DEVS*, SECK and VERBRAECK 2009) and the latter is called DSOL (*Distributed Simulation Object Library*, JACOBS 2005). Figure A.2 shows the relation of DSOL, ESDEVS and user defined DEVS components using ESDEVS.

In DSOL, a simulation event is scheduled as a method execution by a source object on a target object (SECK and VERBRAECK 2009). The library provides components for simulators (i.e., event-scheduling and execution), numerical integrators and probability distributions, etc. ESDEVS implements a parallel DEVS compliant simulator using the event-scheduling simulator in DSOL. It schedules the executions of internal transition functions according to the specified time advance functions, and unschedules them at the reception of external events (except for confluent transition situations).

Dynamic structure DEVS (UHRMACHER 2001) is also implemented by the ESDEVS library so that components and coupling relations can be added and removed dynamically during the simulation run-time. On the whole, the library specifies the meta structure of the DEVS atomic and coupled models, and handles the couplings, output functions and transition functions at an abstract level, so that ESDEVS (together with DSOL) serves as a DEVS simulator.

Following the definition template provided by the ESDEVS library, modelers can specify their own DEVS model components by the inheritance of the `AtomicModel` and `CoupledModel` classes in the library; the two classes are shown in Figure A.3. The separation of concerns between modeling and simulation is guaranteed from the perspective of modelers because simulation related issues such as time management and executions of transition functions are not handled explicitly by modelers (SECK and VERBRAECK 2009) who can concentrate on the model specifications.

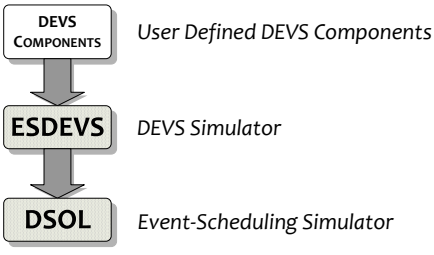


Figure A.2: User defined components and the DSOL and ESDEVS simulators

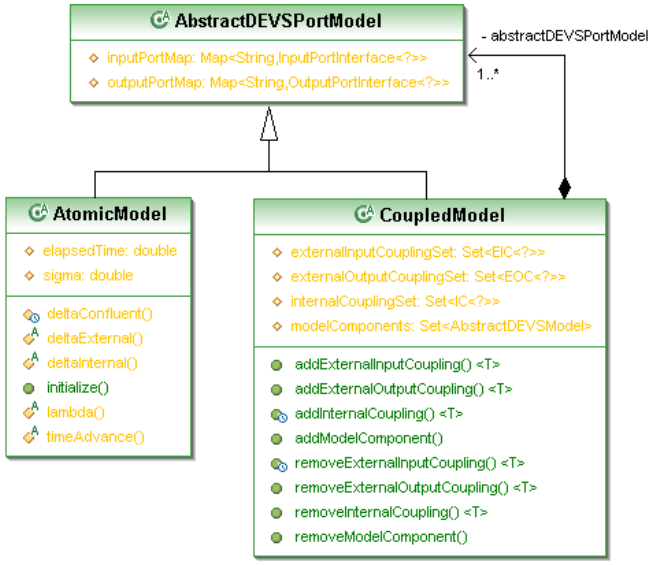


Figure A.3: The atomic and coupled model classes in ESDEVS

B

LIBROS *Library*

B.1 Communication Mechanism

B.1.1 Message Initiators

A vehicle model can be a message initiator, i.e., it can initiate a round of MP. RIE models are message propagators and deliverers. Table B.1 summarizes the possible message types, directions and the triggering conditions of MP in LIBROS.

B.1.2 Message Propagation Rules

1. If a requesting vehicle has a preceding vehicle on the same RIE that it is coupled with, the RIE model shall forward the request message to the preceding vehicle model.
2. If a requesting vehicle's preceding vehicle is not yet found and a RIE model is not coupled with the requesting vehicle, the RIE model shall forward a request message to the vehicle model closest to its start node when there is any vehicle coupled to it.
3. If a requesting vehicle's closest RIE is not yet found, a RIE model shall respond to a request message when the RIE requires or potentially requires a change in movement of the requesting vehicle.
4. A vehicle model shall respond to a request message unconditionally.

Initiator	Type	Direction	Condition
Vehicle	Request	Forward	When the vehicle model does not have information about its next closest RIE of interest or the preceding vehicle.
	Response	Backward	When the vehicle model receives a request message.
	Update	Backward	When the vehicle model changes its acceleration.
Track Segment	Response	Backward	When the track segment receives a request message and the original (vehicle) sender's next closest RIE of interest is not yet found, and the track segment has a different speed limit or the message propagation distance exceeds a predefined value.
3S	Response	Backward	When the 3S model receives a request message and the original (vehicle) sender's next closest RIE of interest is not yet found.
Signal	Update	Backward	When the signal changes its signaling state.

Table B.1: Message initiators

5. After Rules 1. ∼ 3., a RIE model shall propagate a request message:
   (a) when the closest RIE of the requesting vehicle is not found, or
   (b) when the preceding vehicle of the requesting vehicle is not found and when the propagation distance does not exceed a defined bound;
6. A RIE model shall forward a response message:
   (a) to the recipient vehicle model when the message has an addressed recipient that is coupled with the RIE, or
   (b) to the following vehicle when the response message does not have an addressed recipient and the sender is a vehicle which has a following vehicle coupled with the same RIE as the sender, or
   (c) to the vehicle model closest to its end node when there is any vehicle coupled to it and when the message does not have an addressed recipient.
7. If Rule 6. does not apply, a RIE model shall propagate a response message.

B.1.3 Distance Accumulation Rules

1. When a vehicle creates a backward message, the distance contained in the message is set to the value of the vehicle's position on the RIE model with which it is

coupled, deducted by the vehicle's length.

2. When a RIE model receives a backward message, it does not change the value of the distance if the original sender is a vehicle that is coupled to the RIE; otherwise the RIE increases the value by its own length.
3. When a vehicle receives a backward message, its distance to the original sender of the message is the distance contained in the message deducted by its position on the RIE model with which it is coupled.

B.2 Vehicle Model Specification

The vehicle model specification is a structure

$$\texttt{RailVehicle} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta) \tag{B.1}$$

where

$X = \{(p,m) | p \in \{\texttt{I/RIE}\}, m \in X_p\}$ is the set of *input port* and *messages*,

$Y = \{(p,m) | p \in \{\texttt{O/F/RIE}, \texttt{O/B/RIE}\}, m \in Y_p\}$
is the set of *output ports* and *messages*,

$S = S' \times S'' \times I_{NCRI} \times I_{PV}$ is the set of *states*, where

$S' = \{(\rho, a) | \rho \in \{\texttt{START}, \texttt{FOLLOW}, \texttt{MOVE\_TO\_NCRI}, \texttt{DWELL}, \texttt{STOP}\}, a \in \mathbb{R}\}$
is the set of the vehicle's *primary states* consist of its phase and acceleration,

$S'' = \{(v, \hat{v}, M, p, l, d_{total}, \cdots) | 0 \le v \le v_{max} \in \mathbb{R}^+, p, l, d_{total} \in \mathbb{R}_0^+\}$
is the set of *secondary states* that consist of its speed, speed limit, anticipated movement, position on the RIE where it is located, length, total distance traveled, etc.,

$M = (M_0, \cdots, M_j, \cdots, M_k), M_j = (t_j, v_j, d_j, a_j), k \in \mathbb{N} \cup \varnothing, t_j \ne 0, a_{i \in [0,k-1]} \ne a_{i+1}$
is the *anticipated movement* that consists of a number of successive movement segments each of which defined with the time (duration), end speed, moving distance, and acceleration of the segment,

$I_{NCRI} = \{(d_{NCRI}, l_{NCRI}, \cdots) | d_{NCRI}, v_{NCRI} \in \mathbb{R}_0^+\}$
is the set of *information about the NCRI* that consists of the distance, speed limit, etc., of the NCRI,

$I_{PV} = \{(d_{PV}, v_{PV}, a_{PV})\}$ is the set of *information about the PV* that consists of the distance of the PV and its speed and acceleration

$\delta_{ext} : Q \times X^b \to S \backslash S', \tau = 0$ is the external transition function, where

$Q = \{(s,e) \mid s \in S, e \in [0, ta(s)]\}$	is the set of *total states* with the elapsed time since last transition
$\delta_{int} : S \to S$	is the internal transition function
$\delta_{con} := \delta_{ext}(\delta_{int}(s))$	is the confluent transition function
$\lambda : S \to Y^b$	is the output function
$ta : S \to \mathbb{R}_0^+ \cup \infty :=$ Return τ	is the time advance function

B.2.1 External Transition Function

Require: The arrival of an external event m at port `I/RIE`

```
1  function δ_ext(e, m)
2      UPDATESTATE(e)
3      if m is not sent to this vehicle then
4          τ ← τ − e
5          return
6      end if
7      τ ← 0
8      if m is a request message then
9          PREPAREBACKWARDMESSAGE(p − l, sender of m)        ▷ a response message
10     else                                                  ▷ m is a backward message
11         if m is from a vehicle then
12             I_PV ← information contained in m
13         else
14             I_NCRI ← information contained in m
15         end if
16         if ρ is not WAIT or DWELL then
17             M ← COMPUTEMOVEMENT
18         end if
19     end if
20 end function
```

B.2.2 Output Function

```
1  function λ
2      if is finished M_0 then                               ▷ finished one movement segment
3          if M_1 exists then                                ▷ has a next movement segment
4              PREPAREBACKWARDMESSAGE(p + d_0 − l)           ▷ d_0 ∈ M_0, an update message
5              if is to brake approaching an Interlocking then
6                  PREPAREFORWARDMESSAGE(−p + d_0)           ▷ request to access the interlocking
7              end if
8          else                                              ▷ completed one movement
9              if ρ is MOVE_TO_NCRI then
```

```
10                CoupleToNCRI
11                PrepareForwardMessage(0)                ▷ a request message
12                if NCRI is a trailing point then
13                    PrepareBackwardMessage(p + d_0 − l)
14                end if
15            else if ρ is STOP then                      ▷ the vehicle is just stopped
16                PrepareBackwardMessage(p + d_0 − l)
17            else if ρ is START and (I_NCRI or I_PV unknown) then
18                PrepareForwardMessage(−p)
19            end if
20        end if
21    else if is to change a then                         ▷ can only be true when a ≠ a_0 after the δ_ext
22        PrepareBackwardMessage(p − l)
23    end if
24    SendForwardMessage
25    SendBackwardMessage
26 end function
```

```
1  function PrepareForwardMessage(d)
2      create new message m→ with distance d
3      m_f ← true
4  end function
```

```
1  function PrepareBackwardMessage(d, r)
2      create new message m← with distance d and receiver r
3      m_b ← true
4  end function
```

```
1  function SendForwardMessage
2      if m_f is true then
3          send m→ through O/RIE/F
4          m_f ← false
5      end if
6  end function
```

```
1  function SendBackwardMessage
2      if m_b is true then
3          send m← through O/RIE/B
4          m_b ← false
5      end if
6  end function
```

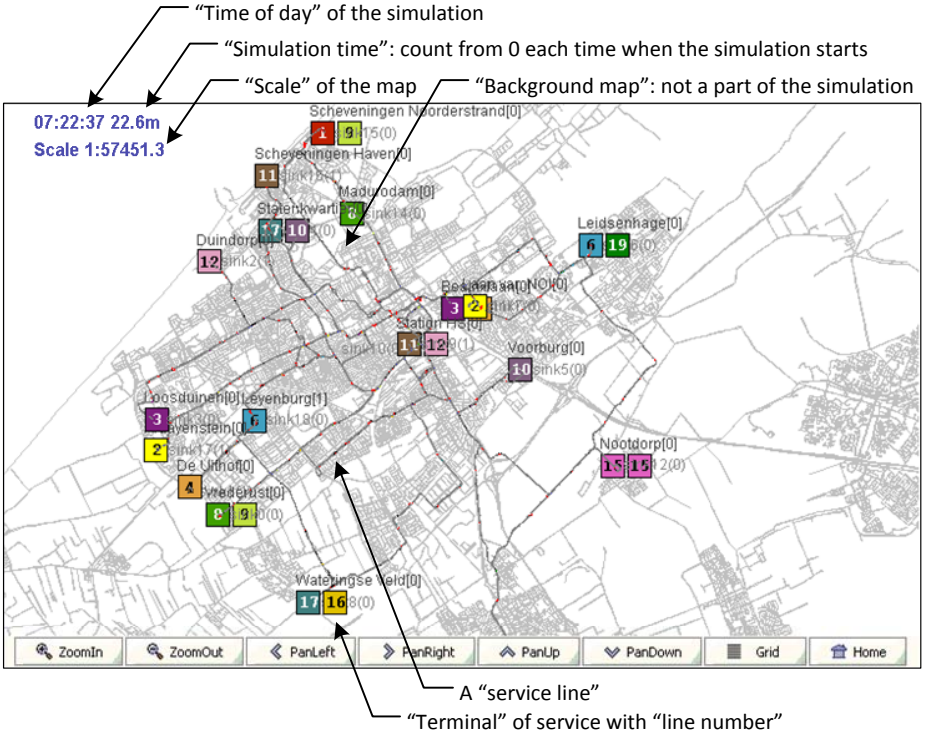
B.2.3 Internal Transition Function

```
1  function δ_int
2      if is finished M_0 then                          ▷ finished one movement segment
3          UPDATESTATE
4          remove M_0 from M                            ▷ by which M_{j≠0} if any becomes M_{j−1}
5          if M_0 exists then                           ▷ has a next movement segment
6              a ← a_0
7              τ ← t_0                                  ▷ continue with the next movement segment
8          else                                         ▷ completed one movement
9              if ρ is MOVE_TO_NCRI then
10                 d_total ← d_total + d_NCRI
11                 p ← 0
12                 if is entered a stopping place then
13                     <do something related to the stopping place>
14                 end if
15                 if is entered a stopping place and is to dwell then
16                     l_NCRI ← 0                       ▷ the vehicle is going to stop
17                 else
18                     I_NCRI ← empty
19                 end if
20             else if ρ is DWELL then                  ▷ the vehicle just finished dwelling
21                 if is to dwell again then            ▷ in case of double halting at the stopping place
22                     ρ ← MOVE_TO_NCRI
23                 else
24                     I_NCRI ← empty
25                 end if
26             end if
27             if ρ is MOVE_TO_NCRI or START or FOLLOW then
28                 if I_NCRI is not empty then
29                     COMPUTEMOVEMENT
30                     τ ← 0                            ▷ continue with the next movement
31                 else
32                     τ ← ∞  ▷ wait for a response message; a request message is already sent
33                 end if
34             else if ρ is STOP then                   ▷ the vehicle just stopped driving
35                 if is at stopping place then         ▷ the vehicle just stopped at the halting place
36                     <do something related to the halting>
37                     t ← calculate dwell time
38                     M_0 ← (t, 0, 0, 0)
39                     ρ ← DWELL
40                     τ ← t
41                 else
42                     τ ← ∞
```

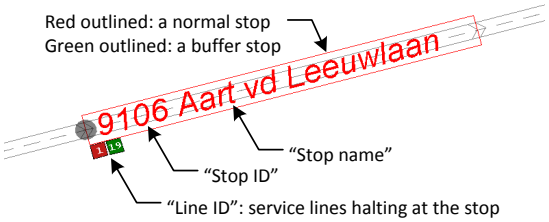
```
            end if
        else if ρ is DWELL then                    ▷ the vehicle just finished dwelling
            M_0 ← (0,0,0,0)
            ρ ← START
            τ ← 0                                  ▷ still need to send a request message
        end if
    end if
else if M is empty then                            ▷ it can be true only after the δ_ext
    τ ← ∞
else                                               ▷ it is true only after the δ_ext when M is not empty
    a ← a_0
    τ ← t_0
end if
end function
```

B.3 Animation Legend

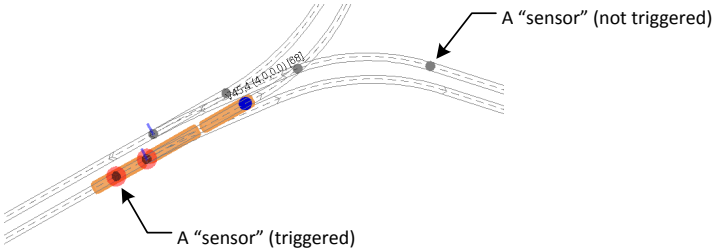
Animation Window



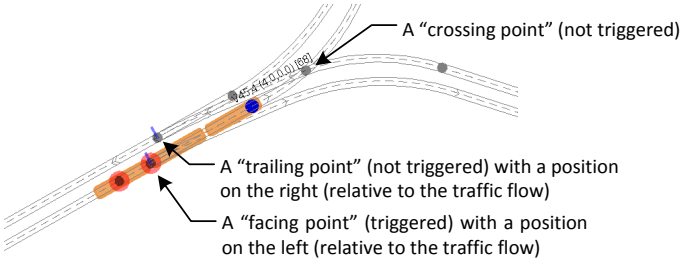
A Stop



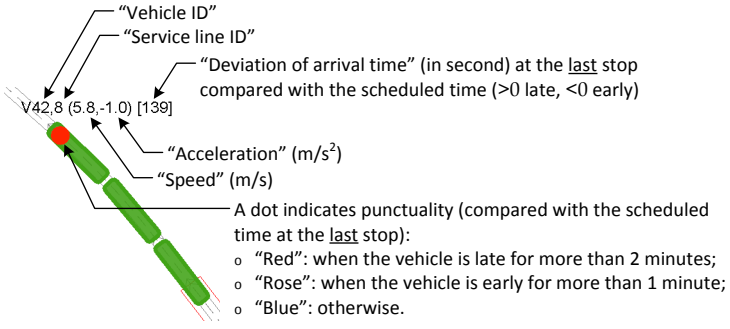
Sensors



Points



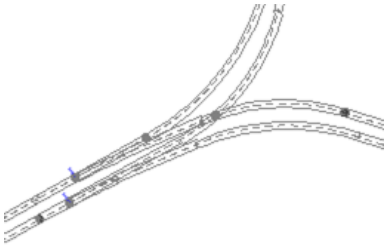
A Vehicle



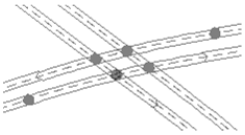
B.4 Infrastructure Composite Examples

Some animation videos can be found at www.youtube.com/shanghaielaine.

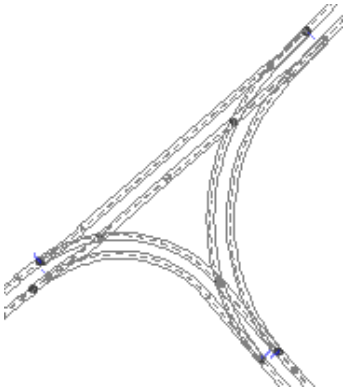
A "Y" Crossing



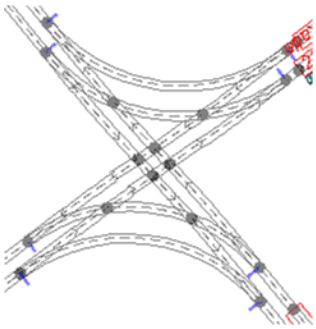
A Quad-Diamond Crossing



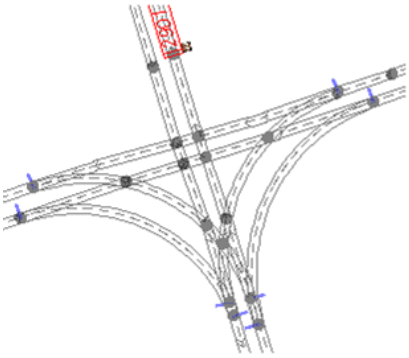
A "T" Crossing



A Butterfly Union



A Half Union



C Model Generation

C.1 Infrastructure CAD Entities

A short description of the CAD Entities – see Figure 3.2 for examples.

Infrastructure Name	CAD Entity	Geometry	Label
Source	A labeled *block* entity named as "source" containing a *circle*	The center of the circle is the location of the source.	Source name
Track	A colored *line* or *arc* entity	The shape and location of the line or arc is the shape and location of the track segment.	–
Stop	A labeled *block* entity named as "stop" containing a *circle*	The center of the circle is the start location of the stop.	Stop ID, name, #halting places (one halting place is 37 meters in length)

The color of tracks represents a speed limit in the following category:

- orange – free lane (50 km/h),
- blue – street track (35 km/h),
- green – city track (15 km/h).

C.2 Infrastructure Composites

Stop Composite It represents one or a number of successive stopping (i.e., halting) placings.

Stop Composite $(1,1)$ e_S		
Composites		$t_1, t_2, \cdots, t_n$
Entry	1. e_{Sen1}	t_1
Exit	1. e_{Sex1}	t_n

Table C.1: Stop Composite

"Y" Composite See § 5.2.3.2 Table 5.1.

"T" Composite See § 5.2.3.2 Table 5.2.

Quad-Diamond Composite

Quad-diamond composite $(4,4)$ e_Q		
Composites		cp_1, cp_2, cp_3, cp_4
Paths	p_1	cp_{1ex2} to cp_{2en1}
	p_2	cp_{2ex2} to cp_{3en1}
	p_3	cp_{3ex2} to cp_{4en1}
	p_4	cp_{4ex2} to cp_{1en1}
Bound		$b \in \mathbb{N}$
Entry	1. e_{Qen1}	cp_{1en2}
	2. e_{Qen2}	cp_{2en2}
	3. e_{Qen3}	cp_{3en2}
	4. e_{Qen4}	cp_{4en2}
Exit	1. e_{Qex1}	cp_{1ex1}
	2. e_{Qex2}	cp_{2ex1}
	3. e_{Qex3}	cp_{3ex1}
	4. e_{Qex4}	cp_{4ex1}

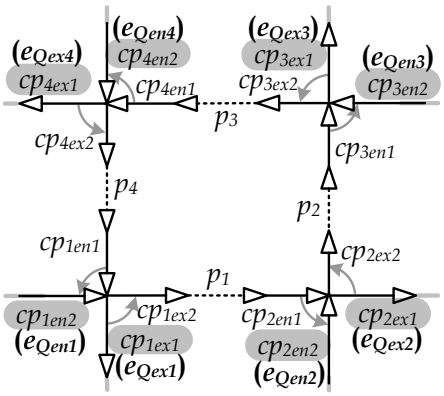
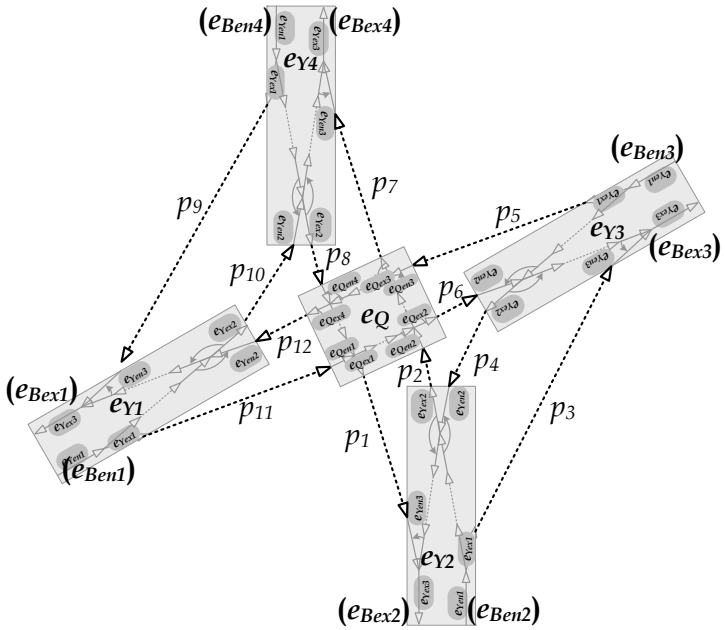


Table C.2: Quad-diamond composite defined as a graph pattern

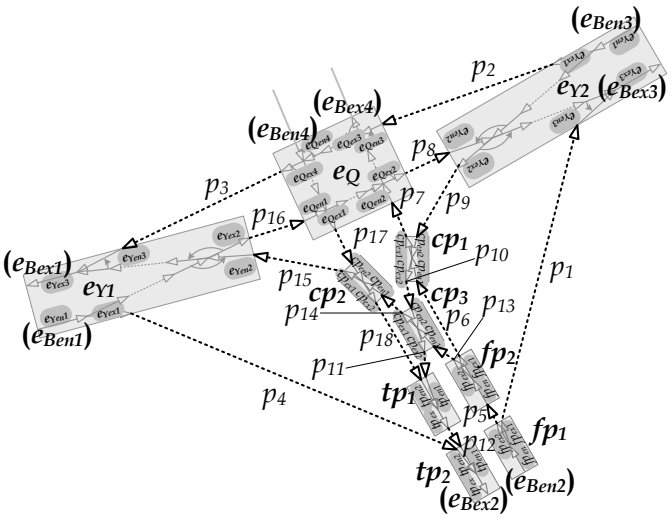
Butterfly Union Composite



Butterfly Union composite $(4,4)$ e_B				
Composites		$e_Q, e_{Y1}, e_{Y2}, e_{Y3}, e_{Y4}$		
Paths	p_1	e_{Qex1} to e_{Y2en3}	p_7	e_{Qex3} to e_{Y4en3}
	p_2	e_{Y2ex2} to e_{Qen2}	p_8	e_{Y4ex2} to e_{Qen4}
	p_3	e_{Y2ex1} to e_{Y3en3}	p_9	e_{Y4ex1} to e_{Y1en3}
	p_4	e_{Y3ex2} to e_{Y2en2}	p_{10}	e_{Y1ex2} to e_{Y4en2}
	p_5	e_{Y3ex1} to e_{Qen3}	p_{11}	e_{Y1ex1} to e_{Qen1}
	p_6	e_{Qex2} to e_{Y3en2}	p_{12}	e_{Qex4} to e_{Y1en3}
Bound		$b \in \mathbb{N}$		
Entry	1. e_{Ben1}	e_{Y1en1}	3. e_{Ben3}	e_{Y3en1}
	2. e_{Ben2}	e_{Y2en1}	4. e_{Ben4}	e_{Y4en1}
Exit	1. e_{Bex1}	e_{Y1ex3}	3. e_{Bex3}	e_{Y3ex3}
	2. e_{Bex2}	e_{Y2ex3}	4. e_{Ben4}	e_{Y4ex3}

Table C.3: Butterfly union composite defined as a graph pattern containing one quad-diamond and four “Y” composites

Half Union Composite



Half Union composite $(4,4)$ e_B				
Composites		$e_Q, e_{Y1}, e_{Y2}, fp_1, fp_2, tp_1, tp_2, cp_1, cp_2, cp_3$		
Paths	p_1	e_{Qex1} to e_{Y2en3}	p_7	e_{Qex3} to e_{Y4en3}
	p_2	e_{Y2ex2} to e_{Qen2}	p_8	e_{Y4ex2} to e_{Qen4}
	p_3	e_{Y2ex1} to e_{Y3en3}	p_9	e_{Y4ex1} to e_{Y1en3}
	p_4	e_{Y3ex2} to e_{Y2en2}	p_{10}	e_{Y1ex2} to e_{Y4en2}
	p_5	e_{Y3ex1} to e_{Qen3}	p_{11}	e_{Y1ex1} to e_{Qen1}
	p_6	e_{Qex2} to e_{Y3en2}	p_{12}	e_{Qex4} to e_{Y1en3}
	p_6	e_{Qex2} to e_{Y3en2}	p_{12}	e_{Qex4} to e_{Y1en3}
	p_6	e_{Qex2} to e_{Y3en2}	p_{12}	e_{Qex4} to e_{Y1en3}
	p_6	e_{Qex2} to e_{Y3en2}	p_{12}	e_{Qex4} to e_{Y1en3}
Bound		$b \in \mathbb{N}$		
Entry	1. e_{Ben1}	e_{Y1en1}	3. e_{Ben3}	e_{Y3en1}
	2. e_{Ben2}	e_{Y2en1}	4. e_{Ben4}	e_{Y4en1}
Exit	1. e_{Bex1}	e_{Y1ex3}	3. e_{Bex3}	e_{Y3ex3}
	2. e_{Bex2}	e_{Y2ex3}	4. e_{Ben4}	e_{Y4ex3}

Table C.4: Half union composite defined as a graph pattern containing one quad-diamond and two “Y” composites and seven points

Facing Turnout Composite It represents a (diverging) turnout that allows a facing-point movement.

Facing Turnout composite $(1,2)$ e_F		
Composites		$fp_{en}, fp_{ex1}, fp_{ex2}$
Entry	1. e_{Fen1}	fp_{en}
Exit	1. e_{Fex1}	fp_{ex1}
	2. e_{Fex2}	fp_{ex2}

Table C.5: Facing Turnout Composite

Trailing Turnout Composite It represents a (converging) turnout for a trailing-point movement.

Trailing Turnout composite $(2,1)$ e_{TR}		
Composites		$tp_{en1}, tp_{en2}, tp_{ex}$
Entry	1. e_{Ten1}	tp_{en1}
	2. e_{Ten2}	tp_{en2}
Exit	1. e_{Tex1}	tp_{ex}

Table C.6: Trailing Turnout Composite

Diamond Composite It represents of a fixed diamond (crossing).

Diamond composite $(2,2)$ e_C		
Composites		$cp_{en1}, cp_{en2}, cp_{ex1}, cp_{ex2}$
Entry	1. e_{Cen1}	cp_{en1}
	2. e_{Cen2}	cp_{en2}
Exit	1. e_{Cex1}	cp_{ex1}
	2. e_{Cex2}	cp_{ex2}

Table C.7: Diamond Composite

Misc Composite

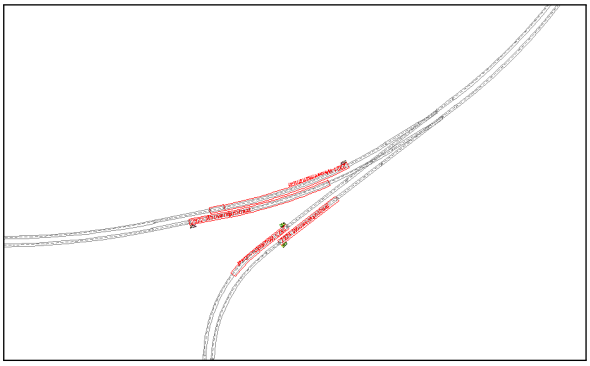
Misc composite (m, n) e_M		
Composites		$e_{F1}, e_{F2}, \cdots, e_{F,a}$, $e_{TR1}, e_{TR2}, \cdots, e_{TR,b}$, $e_{C1}, e_{C2}, \cdots, e_{C,c}$
Paths	p_1	$\cdots$ to $\cdots$
	p_2	$\cdots$ to $\cdots$
	$\cdots$	$\cdots$ to $\cdots$
	p_d	$\cdots$ to $\cdots$
Bound		$b \in \mathbb{N}$
Entry	1. e_{Men1}	$\cdots$
	2. e_{Men2}	$\cdots$
	$\cdots$	$\cdots$
	m. $e_{Men,m}$	$\cdots$
Exit	1. e_{Mex1}	$\cdots$
	2. e_{Mex2}	$\cdots$
	$\cdots$	$\cdots$
	n. $e_{Mex,n}$	$\cdots$

Table C.8: Misc composite

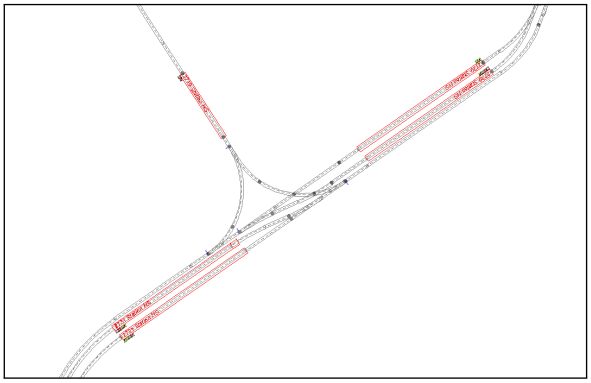
C.3 Infrastructure Model Examples

Some animation videos can be found at www.youtube.com/shanghaielaine.

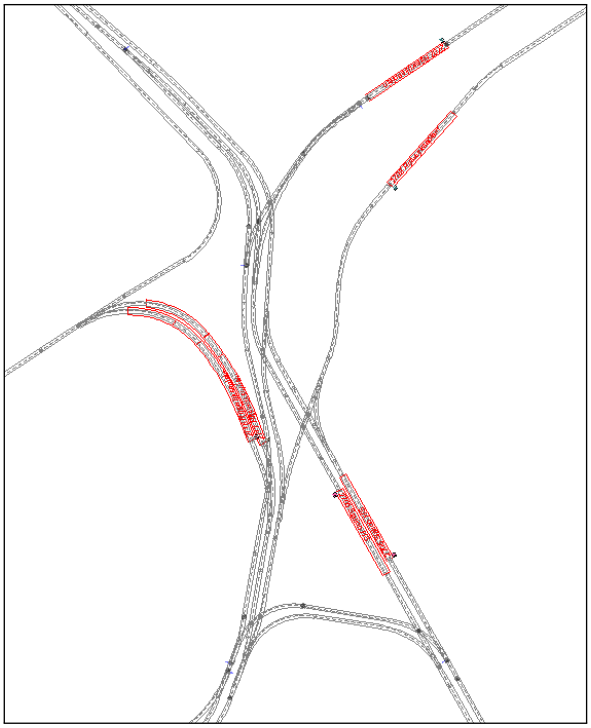
Wouwermanstraat



Station HS

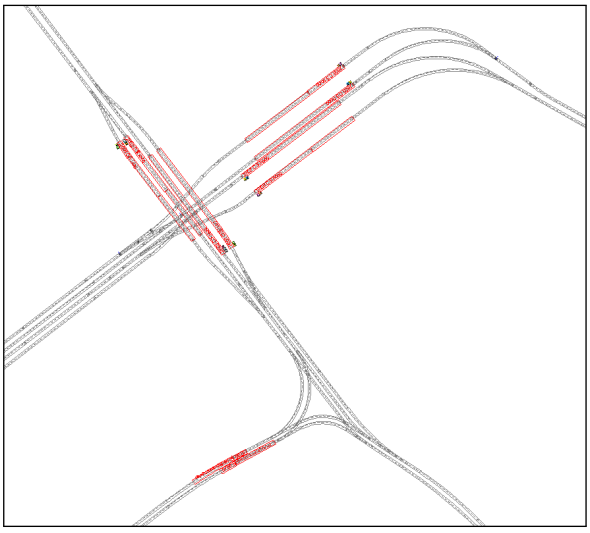


Rijswijkseplein



Bierkade

Station CS



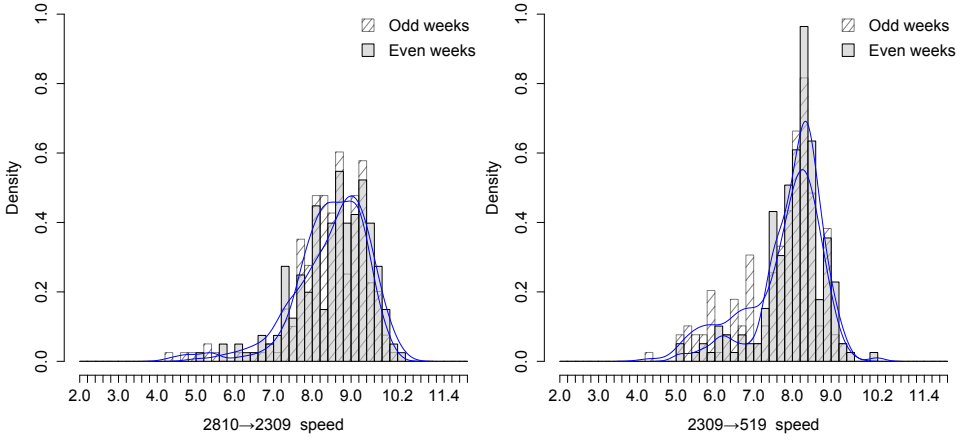
D

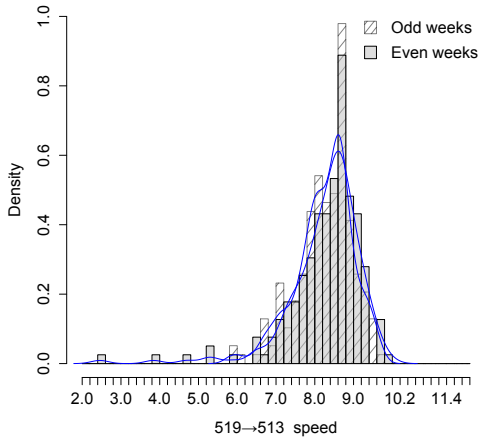
Model Validation

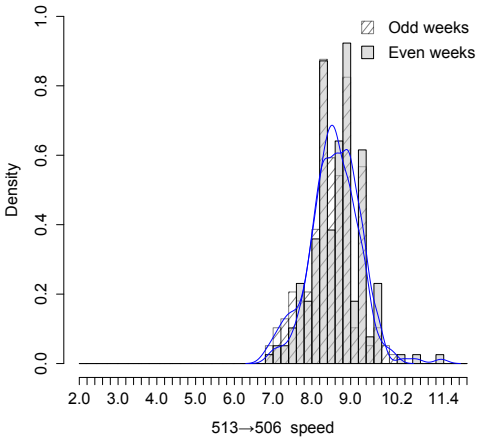
D.1 Validation of Calibration Results

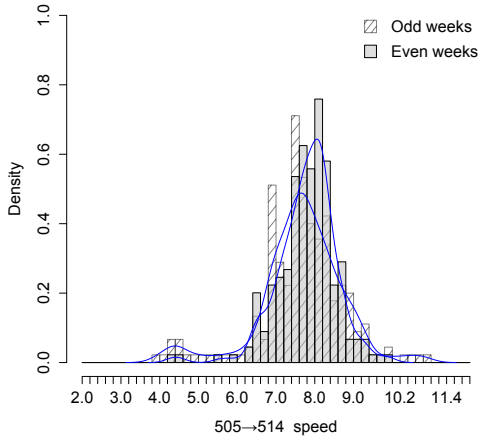
D.1.1 Calibration vs. Validation Datasets

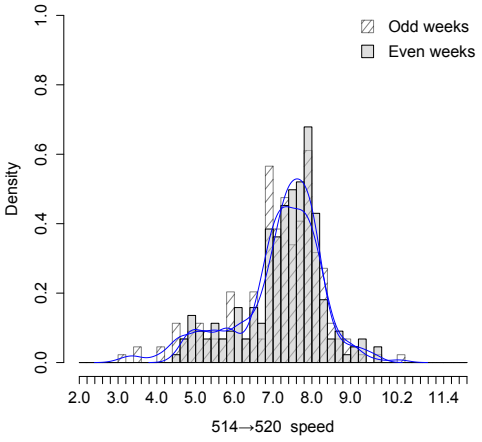
The density functions of the driving speeds (in m/s) from the EBS data per stop-to-stop interval used in the test case in § 6.4.3: the odd weeks vs. the even weeks of October 2011. The datasets from the odd weeks are used for the calibration and the datasets from the even weeks are used for the validation.

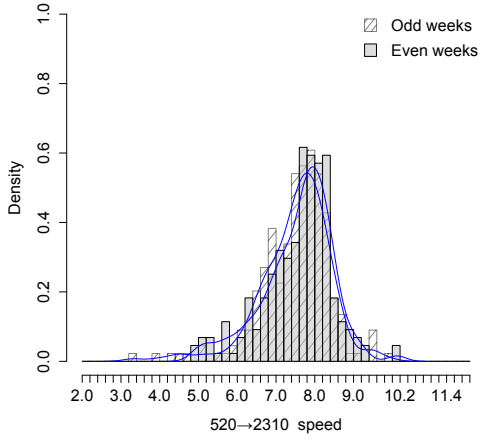


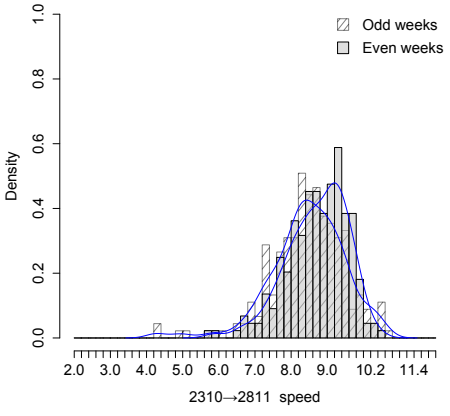

Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
519→513 speed


Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
513→506 speed


Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
505→514 speed

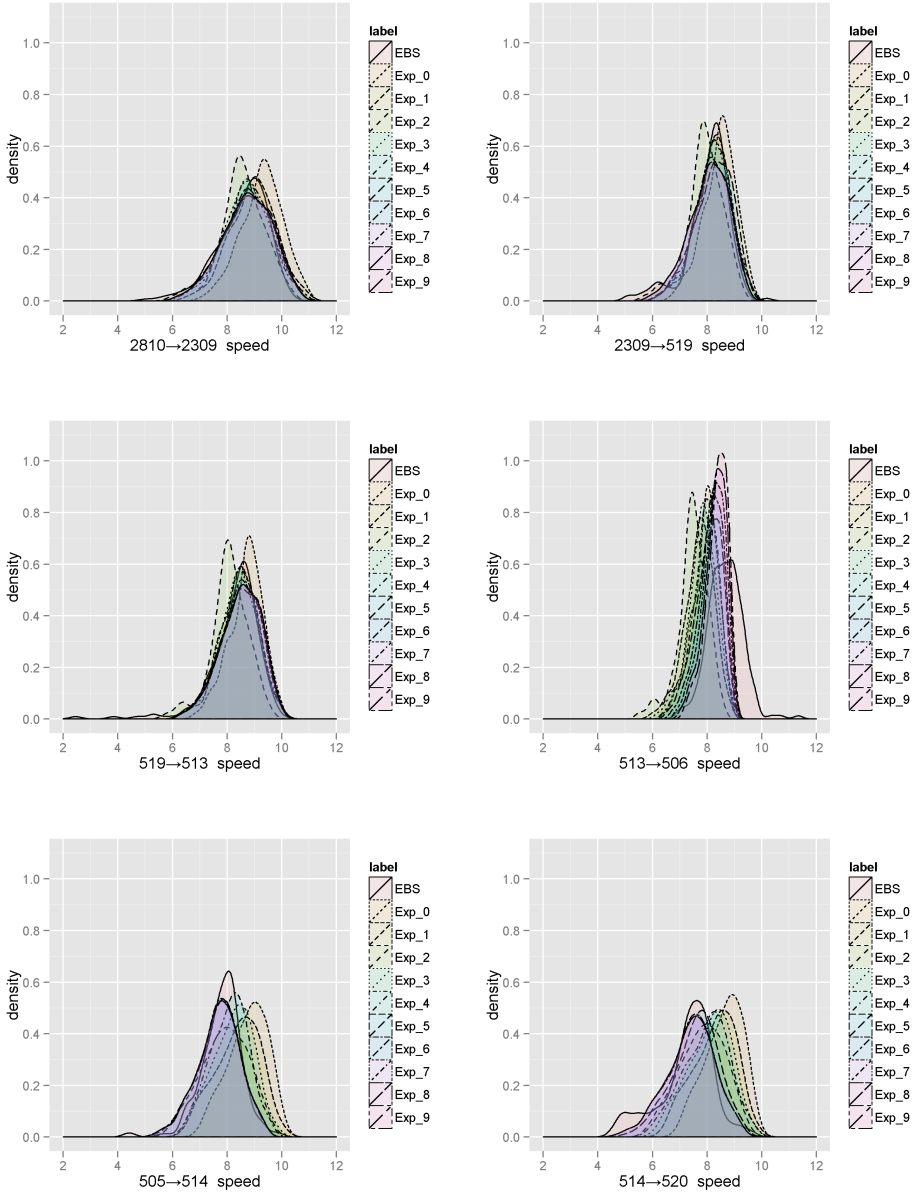

Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
514→520 speed

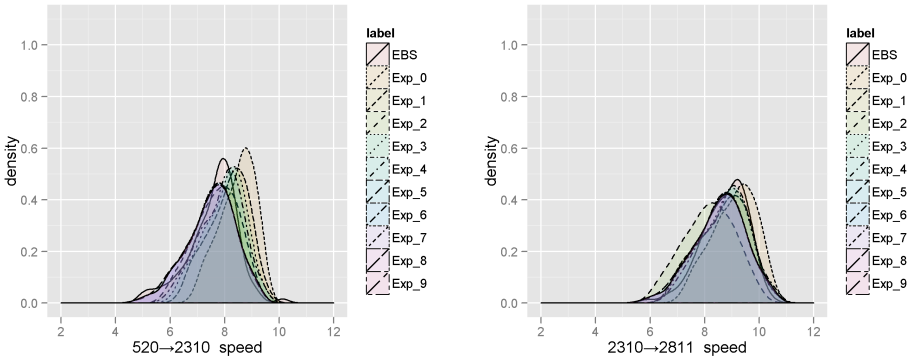

Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
520→2310 speed


Odd weeks
Even weeks
Density
0.0 0.2 0.4 0.6 0.8 1.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.2 11.4
2310→2811 speed

D.1.2 Calibration Results

The density functions of the driving speeds (in m/s) from the model calibration experiments in § 6.4.3 vs. the EBS data.





D.2 Expertise of Panelists

The nine experts participated in the LIBROS model validation have the following expertise related to public transport.

1. An expert in public transport network design and timetable optimization, with special interests in service reliability.
2. An expert in transport planning and optimization of the planning process and public transport economics.
3. An expert in design and development of public transport networks and public transport economics.
4. An expert in public transport planning and scheduling (vehicles, timetables and duties), working conditions, finance, transport development, trip-time and passenger analysis.
5. An expert in the EBS database system. The system contains information about the operational status of vehicles and the passenger information. The expert is also responsible for the central information system used by the dispatchers.
6. An expert in transport planning and optimization of light rail operations.
7. An expert in operations of the public rail and light-rail transport systems.
8. An expert in light-rail transport systems design and maintenance, rail-wheel interface, safety issues and exploitation issues.
9. An expert in strategic urban transport network and schedule design based on transport modeling.

D.3 Validation Questionnaire

This questionnaire is meant to be answered with your observations of the provided model animation videos. An animation legend is provided to explain the visualization.

The time reference in the questions refer to the "time of day" of the simulation (at the up left corner of the animation window). The animation speed in the videos is twice the real time, except for the ones explicitly clarified as ten times the real time in the videos. You may need to pause and replay the videos in order to answer the questions. We thank you in advance for your time, patience and support.

Location I. Wouwermanstraat

1. A line 12 vehicle (V49 pink) is halting at stop 2922 at 07:38. When the second vehicle (line 11 V48 brown) arrives at this stop, are the second vehicle's decelerating behavior and stopping position correct?
2. When the vehicle (line 8 V65 green) at stop 2924 leaves the stop and drives through the converging tracks, are the two sensors triggered and released on time?
3. At 07:39 when the two vehicles depart from stop 2922 one after the other:
   (a) Is the converging switch correctly pushed by the first vehicle?
   (b) Does the second vehicle halt and drive according to the organization's regulations?
4. At 07:53 when the two vehicles depart at from stop 2921, do they drive according to the organization's regulations? Please specify the errors if they are not mentioned be-fore.
5. In this video clip, do the vehicles arrive and depart stops in a realistic manner in terms of their speed, acceleration and deceleration? Please specify the errors if they are not mentioned before.
6. In this video clip, are there any abnormal behaviors of the vehicles, sensors or switches, or in any other aspects besides the ones already motioned before?

Location II. Station HS

7. At stop 2730, a line 17 vehicle is halting (V203 dark green). A line 16 vehicle (V181 gold) enters the scene at 08:30 approaching the stop. Before V181 stops behind V203, V203 starts driving. Is the cruising and following movement of V181 correctly modeled?
8. At 08:45 the line 8 vehicle (V238 green) at stop 2720 and the line 17 vehicle (V240 dark green) at stop 2730 depart at about the same time. When they drive towards the converging tracks, V238 passes first.
   (a) Is V240's waiting position correct?
   (b) In such cases, does the waiting position affect the simulation output such as waiting time and total trip time?
9. At stop 2721, the line 12 vehicle (V270 pink) and the line 8 vehicle (V271 green) are halting. When they drive towards the converging tracks successively:

(a) Is V271's waiting position correct?

(b) Shall V271 wait until V270 passes the intersection in this case?

10. At 08:50 the line 18 vehicle (V275 dark red) at stop 2731 departs from stop 2730 and drives towards the converging tracks. A line 11 vehicle (V278 brown) is at this intersection.

    (a) Is V275's waiting position correct?

    (b) In such cases, does the waiting position affect the simulation output such as waiting time and total trip time?

11. In this video clip, do the vehicles pass the intersections according to safety measures? Please specify the errors if they are not mentioned before.
12. In this video clip, do the vehicles arrive and depart stops in a realistic manner in terms of their speed, acceleration and deceleration? Please specify the errors if they are not mentioned before.
13. In this video clip, are there any abnormal behaviors of the vehicles, sensors or switches, or in any other aspects besides the ones already motioned before?

Location III. Rijswijkseplein

14. At stop 2707 a line 17 vehicle (V2 dark green) is halting. Short before the vehicle starts driving, the switch in front of the vehicle turned from left to right (relative to the vehicle's position).

    (a) Does the switch turn at the about right time?

    (b) Are the two sensors triggered and released on time?

15. At 07:29, a line 17 vehicle (V47 dark green) enters the scene and drives over a diverging switch. Soon after V47, a line 9 vehicle (V49 green) approaches the same switch. The switch turned from right to left (relative to the vehicle's position).

    (a) Does the switch turn at about the right time?

    (b) Are the sensors triggered and released on time?

16. In this video clip, do the vehicles pass the intersections according to safety measures? Please specify the errors if they are not mentioned before.
17. In this video clip, are there any abnormal behaviors of the vehicles, sensors or switches, or in any other aspects besides the ones already motioned before?

Location IV. Bierkade

18. At 07:31 a line 10 vehicle (V46 grey) enters the intersection. A line 15 vehicle (V59 pink) at stop 2852 is about to depart. Is V59's waiting position correct?
19. When V46 arrives at stop 2851, a line 1 vehicle (dark red) starts to leave. Is V46's cruising and following movement correct?

20. After 07:32 a line 10 vehicle (V58 grey) and a line 16 vehicle (V63 gold) arrives and departs stop 2852 successively. V58 turns right and V63 drives straight.
    (a) Does the switch turn at about the right time?
    (b) Are the sensors triggered and released on time?
21. In this video clip, do the vehicles pass the intersection according to safety measures? Please specify the errors if they are not mentioned before.
22. In this video clip, do the vehicles arrive and depart stops in a realistic manner in terms of their speed, acceleration and deceleration? Please specify the errors if they are not mentioned before.
23. In this video clip, are there any abnormal behaviors of the vehicles, sensors or switches, or in any other aspects besides the ones already motioned before?

Location V. Station CS

24. In this video clip, do the vehicles pass the intersections according to safety measures? Please specify the errors if they are not mentioned before.
25. In this video clip, do the vehicles arrive and depart stops in a realistic manner in terms of their speed, acceleration and deceleration? Please specify the errors if they are not mentioned before.
26. In this video clip, are there any abnormal behaviors of the vehicles, sensors or switches, or in any other aspects besides the ones already motioned before?

References

ACHERMANN, F., M. LUMPE, J.-G. SCHNEIDER and O. NIERSTRASZ (2001). PICCOLA – a Small Composition Language. *Formal Methods for Distributed Processing – A Survery of Object-Oriented Appraoches*. Ed. by H. Bowman and J. Derrick. Cambridge University Press, pp. 403–426 (cit. on p. 25).

ACKOFF, R. L. (1989). From data to wisdom. *Journal of Applied Systems Analysis*, 16, pp. 3–9 (cit. on p. 44).

ACKOFF, R. L. (1973). Science in the Systems Age: Beyond IE, OR, and MS. *Operations Research*, 21(3), pp. 661–671 (cit. on p. 12).

ACKOFF, R. L. (1978). *The Art of Problem Solving: accompanied by Ackoff's Fables*. John Wiley & Sons (cit. on p. 13).

ALEXOPOULOS, C. and A. F. SEILA (1998). Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice. Ed. by J. Banks. New York: John Wiley. Chap. Output data analysis (cit. on pp. 177–179).

ALSTRUP, S., C. GAVOILLE, H. KAPLAN and T. RAUHE (2004). Nearest Common Ancestors: A Survey and a New Algorithm for a Distributed Environment. *Theory of Computing Systems*, 37 (3), pp. 441–456 (cit. on p. 77).

AMSTERDAM, J. (1993). Automated Qualitative Modeling of Dynamic Physical Systems. PhD thesis. Artifical Intelligence Laboratory, Massachusetts Institue of Technology (cit. on p. 2).

ANDERSON, T. W. and D. A. DARLING (1952). Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes. *Annals of Mathematical Statistics*, 23, pp. 193 –212 (cit. on p. 195).

ANDERSSON, P. (2006). *Hyperedge Replacement Grammars*. Final versions of lecture notes, Formal Languages, Department of Computing Science, Umeå University, Sweden (cit. on p. 134).

ARTHUR, J. D. and R. E. NANCE (2007). Investigating the use of software requirements engineering techniques in simulation modelling. *Journal of Simulation*, 1, pp. 159–174 (cit. on p. 176).

ASZTALOS, M., E. SYRIANI, M. WIMMER and M. KESSENTINI (2011). Towards Transformation Rule Composition. *Electronic Communications of the EASST - Proceedings of the 4th International Workshop on Multi-Paradigm Modeling*. 42 (cit. on pp. 55, 56, 154).

ATKINSON, C. and T. KÜHNE (2003). Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5), pp. 36–41 (cit. on p. 22).

BALCI, O. (1998). Verification, Validation, and Testing. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Ed. by J. Banks. Wiley Interscience, pp. 335–393 (cit. on pp. 57, 176, 202).

BALCI, O. (2004). Quality assessment, verification, and validation of modeling and simulation applications. *Proceedings of the 2004 Winter Simulation Conference*. Vol. 1, pp. 122 –129 (cit. on p. 64).

BALCI, O. and R. E. NANCE (1987). Simulation model development environments: a research prototype. *Journal of the Operational Research Society*, 38(8), pp. 753–763 (cit. on p. 175).
BALCI, O. (2012). A life cycle for modeling and simulation. *Simulation*, 88(7), pp. 870–883 (cit. on pp. 15, 176).
BALLOU, D. P. and H. L. PAZER (1985). Modeling Data and Process Quality in Multi-Input, Multi-Output Information Systems. *Management Science*, 31(2), pp. 150 –162 (cit. on p. 46).
BALOGH, A. and D. VARRÓ (2006). Pattern Composition in Graph Transformation Rules. *European Workshop on Composition of Model Transformations* (cit. on pp. 56, 154).
BANKS, J., ed. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Wiley Interscience (cit. on pp. 62–64, 122).
BANKS, J., J. S. CARSON II, B. L. NELSON and D. M. NICOL (2010). *Discrete-Event System Simulation*. 5th. Pearson Education (cit. on pp. 1, 2, 62, 63, 175, 177–179, 198, 201).
BARROS, F. J. (1997). Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation*, 7(4), pp. 501 –515 (cit. on p. 112).
BATINI, C. and M. SCANNAPIECO (2006). *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer-Verlag Berlin Heidelberg (cit. on pp. 47–49).
BATINI, C., C. CAPPIELLO, C. FRANCALANCI and A. MAURINO (2009). Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, 41(3), 16:1 –16:52 (cit. on pp. 45–50).
BEN-ARI, M. (2012). *Mathematical Logic for Computer Science*. Springer-Verlag London (cit. on p. 127).
BENDFELDT, J.-P., U. MOHR and L. MÜLLER (2000). RailSys, a system to plan future railway needs. *Advances in Transport*, 7, pp. 249–255 (cit. on p. 60).
BERGE, C. (1973). *Graphs and Hypergraphs*. Translation and revised edition of Graphes et Hypergraphes 1970. North-Holland Publishing Company (cit. on p. 127).
BERGE, C. (1989). *Hypergraphs: Combinatorics of Finite Sets*. Vol. 45. North-Holland Mathematical Library. Elsevier Science Publishers (cit. on pp. 127, 128).
BERGMANN, G., I. RÁTH, T. SZABÓ, P. TORRINI and D. VARRÓ (2012). Incremental pattern matching for the efficient computation of transitive closure. *Graph Transformations*. Springer Berlin Heidelberg, pp. 386 –400 (cit. on p. 154).
BERGMANN, S. and S. STRASSBURGER (2010). Challenges for the Automatic Generation of Simulation Models for Production Systems. *Proceedings of the 2010 Summer Simulation Multiconference*. Ottawa, Canada, pp. 545–549 (cit. on p. 2).
BLANCHARD, B. S. and W. J. FABRYCKY (2011). *Systems Engineering and Analysis*. 5th. Prentice Hall (cit. on p. 8).
BOEHM, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, pp. 61–72 (cit. on p. 8).
BOVEE, M., R. P. SRIVASTAVA and B. MAK (2003). A conceptual framework and belief-function approach to assessing overall information quality. *International Journal of Intelligent Systems*, 18(1), pp. 51 –74 (cit. on pp. 47, 48, 50).
BRAUDE, E. J. and M. E. BERNSTEIN (2010). *Software Engineering: Modern Approaches*. 2nd. John Wiley & Sons (cit. on pp. 59, 61, 64).
BRAUSE, R. (2004). Model selection and adaptation for biochemical pathways. *Lecture Notes in Computer Science*, 3337, pp. 439–449 (cit. on pp. 3, 4).
BREEDVELD, P. (2009). Modeling and Control of Complex Physical Systems - The Port-Hamiltonian Approach. Ed. by V. Duindam, A. Macchelli, S. Stramigioli and H. Bruyninckx. Springer Verlag. Chap. Port-Based Modeling of Dynamic Systems, pp. 1–52 (cit. on pp. 28, 66).

BRUNI, R., F. GADDUCCI and A. LLUCH LAFUENTE (2010). An Algebra of Hierarchical Graphs. *Trustworthly Global Computing*. Ed. by M. Wirsing, M. Hofmann and A. Rauschmayer. Vol. 6084. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 205 –221 (cit. on p. 127).
BUSATTO, G. and B. HOFFMANN (2001). Comparing notions of hierarchical graph transformation. *Electronic Notes in Theoretical Computer Science*, 50(3), pp. 310 –317 (cit. on p. 127).
COBP (2002). *NATO code of best practice for command and control assessment*. DoD Command and Control Research Program (CCRP). SAS-026 (cit. on p. 4).
CACUCI, D. and M. IONESCU-BUJOR (2010). Sensitivity and Uncertainty Analysis, Data Assimilation, and Predictive Best-Estimate Model Calibration. *Handbook of Nuclear Engineering*. Ed. by C. D. Springer-Verlag Berlin Heidelberg (cit. on p. 176).
CAI, J. (2011). Assessing The Impact of Capacity of Depots and Vehicle Schedule in Transportation Systems. MA thesis. Delft University Of Technology, Faculty Of Technology, Policy And Management (cit. on pp. 3, 218).
CAO, Y., Y. LIU, H. FAN and B. FAN (2012). SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics. *CAD Computer Aided Design*. Article in Press (cit. on p. 2).
CAREY, M. and S. CARVILLE (2002). Testing schedule performance and reliability for train stations. *Journal of the Operational Research Society*, 51(6), pp. 666–682 (cit. on pp. 60, 61).
CAREY, M. and S. CARVILLE (2003). Scheduling and platforming trains at busy complex stations. *Transportation Research Part A: Policy and Practice*, 37(3), pp. 195–224 (cit. on pp. 60, 61).
CAREY, M. and D. LOCKWOOD (1995). A Model, Algorithms and Strategy for Train Pathing. *The Journal of the Operational Research Society*, 46(8), pp. 988–1005 (cit. on p. 60).
CATARCI, T. and M. SCANNAPIECO (2002). Data Quality under the Computer Science Perspective. *Archivi & Computer*, 2 (cit. on p. 48).
CETINKAYA, D. and A. VERBRAECK (2011). Metamodeling and Model Transformations in Modeling and Simulation. *Proceedings of the 2011 Winter Simulation Conference*. Ed. by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White and M. Fu. Phoenix, AZ: IEEE, pp. 3048–3058 (cit. on p. 22).
CHECKLAND, P. (1981). *Systems Thinking, Systems Practice*. John Wiley & Sons (cit. on p. 12).
CHECKLAND, P. (1999). *Systems Thinking, Systems Practice: Includes a 30-Year Retrospective*. John Wiley & Sons (cit. on p. 13).
CHECKLAND, P. and S. HOLWELL (1998). *Information, Systems and Information Systems - making sense of the field*. John Wiley & Sons (cit. on p. 44).
CHEN, Y., M. THURLEY and M. WEYER (2008). Understanding the Complexity of Induced Subgraph Isomorphisms. *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*. Reykjavik, Iceland: Springer-Verlag, pp. 587 –596 (cit. on p. 131).
CHENG, C.-T., X.-Y. WU and K. W. CHAU (2005). Multiple criteria rainfall–runoff model calibration using a parallel genetic algorithm in a cluster of computers. *Hydrological Sciences–Journal–des Sciences Hydrologiques*, 50(6), pp. 1069 –1087 (cit. on p. 181).
CHENG, C.-T., M.-Y. ZHAO, K. W. CHAU and X.-Y. WU (2006). Using genetic algorithm and TOPSIS for Xinanjiang model calibration with a single procedure. *Journal of Hydrology*, 316(1–4), pp. 129 –140 (cit. on p. 176).
CHIN, R. (2007). Mainport Planning Suite: Software Services to Support Mainpoart Planning. PhD thesis. Delft University of Technology (cit. on p. 63).
CLARK, A. (1984). *Elements of Abstract Algebra*. Courier Dover Publications (cit. on p. 224).
CONOVER, W. J. (1971). *Practical Nonparametric Statistics*. New York: John Wiley & Sons (cit. on p. 187).

COOLEY, T. F. (1997). Calibrated models. *Oxford Review of Economic Policy*, 13(3), pp. 55–69 (cit. on pp. 195, 198).
CORMEN, T. H., C. E. LEISERSON, R. L. RIVEST and C. STEIN (2001). *Introduction to Algorithms*. 3rd. MIT Press and McGraw-Hill (cit. on pp. 76, 138, 139, 150, 157, 159, 160).
CORNÉLIO, M. L. (2004). Refactorings as Formal Refinements. PhD thesis. Universidade Federal de Pernambuco, Centro de Informática (cit. on p. 54).
CORRADINI, A., U. MONTANARI, F. ROSSI, H. EHRIG, R. HECKEL and M. LÖWE (1997). Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. *Handbook of Graph Grammars and computing by graph transformation*. Ed. by G. Rozenberg. Vol. 1: Foundations. World Scientific Publishing, pp. 163 –246 (cit. on pp. 129–132).
CRAIK, K. (1943). *The Nature of Explanation*. Cambridge University Press (cit. on p. 12).
CRESWELL, J. W. (2003). *Research design: Qualitative, quantitative, and mixed method approaches*. 2nd. Sage Publications (cit. on pp. 6, 9).
CRNKOVIC, I., J. STAFFORD and C. SZYPERSKI (2011). Software Components beyond Programming: From Routines to Services. *IEEE Software*, 28(3), pp. 22 –26 (cit. on pp. 61, 64, 212).
CROSBIE, R. E. (2010). Grand Challenges in Modeling and Simulation. *SCS M&S Magazine*, 1(1), pp. 1–8 (cit. on pp. 1, 211).
CROSSAN, F. (2003). Research philosophy: towards an understanding. *Nurse Researcher*, 11(1), pp. 46–55 (cit. on pp. 6, 7).
CZARNECKI, K. and U. EISENECKER (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley (cit. on pp. 54–56).
CZARNECKI, K. and S. HELSEN (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), pp. 621 –645 (cit. on pp. 53–56).
DE LARA, J., H. VANGHELUWE and M. ALFONSECA (2004). Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. *Software and Systems Modeling*, 3(3), pp. 194 –209 (cit. on p. 29).
DECROP, A. (2006). *Vacation Decision Making*. CABI Publishing Series. CABI (cit. on p. 7).
DIJKSTRA, E. W. (1982). On the role of scientific thought, 1974. *Selected Writings on Computing: A Personal Perspective*. Springer, pp. 60 –66 (cit. on pp. 64, 66).
DOWLING, R, A SKABARDONIS, J HALKIAS, G MCHALE and G ZAMMIT (2004). Guidelines For Calibration Of Microsimulation Models: Framework And Applications. *Transportation Research Record: Journal of the Transportation Research Board*, (1876), pp. 1–9 (cit. on p. 181).
DREWES, F., H.-J. KREOWSKI and A. HABEL (1997). Hyperedge Replacement Graph Grammars. *Handbook of Graph Grammars and Computing by Graph Transformation*. Ed. by G. Rozenberg. Vol. 1: Foundations. World Scientific Publishing, pp. 95 –162 (cit. on pp. 127, 129, 133, 134).
DREWES, F., B. HOFFMANN and D. PLUMP (2002). Hierarchical Graph Transformation. *Journal of Computer and System Sciences*, 64(2), pp. 249 –283 (cit. on p. 127).
DUBIN, R. (1978). *Theory Building*. revised edition. London: Free Press (cit. on p. 36).
EASTERBY-SMITH, M., R. THORPE and A. LOWE (2001). *Management Research: An Introduction*. 2nd. SAGE Series in Management Research. Sage Publications (cit. on p. 6).
ECKHARDT, K., N. FOHRER and H.-G. FREDE (2005). Automatic model calibration. *Hydrological Processes*, 19(3), pp. 651 –658 (cit. on p. 176).
EECKELAERT, T., W. DAEMS, G. GIELEN and W. SANSEN (2004). Generalized simulation-based posynomial model generation for analog integrated circuits. *Analog Integrated Circuits and Signal Processing*, 40(3), pp. 193–203 (cit. on p. 2).

EHRIG, H., R. HECKEL, M. LÖWE, L. RIBEIRO, A. WAGNER and A. CORRADINI (1997). Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach. *Handbook of Graph Grammars and computing by graph transformation*. Ed. by G. Rozenberg. Vol. 1: Foundations. World Scientific Publishing, pp. 247 – 312 (cit. on p. 132).

EHRIG, H., K. EHRIG, U. PRANGE and G. TAENTZER (2006a). Fundamental Theory for Typed Attributed Graphs and Graph Transformation based on Adhesive HLR Categories. *Fundamenta Informaticae*, 74(1), pp. 31 –61 (cit. on p. 126).

EHRIG, H., K. EHRIG, U. PRANGE and G. TAENTZER (2006b). *Fundamentals of Algebraic Graph Transformation*. Ed. by W. Brauer, G. Rozenberg and A. Salomaa. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag Berlin Heidelberg (cit. on pp. 126, 129, 130).

EOM (2011). *Encyclopedia of Mathematics*. www.encyclopediaofmath.org, Springer & European Mathematical Society (cit. on p. 26).

EPPSTEIN, D. (1999). Subgraph isomorphism in planar graphs and related problem. *Journal of Graph Algorithms and Applications*, 3(3), pp. 1–27 (cit. on p. 131).

ESTEFAN, J. (2008). *Survey of Model-Based Systems Engineering (MBSE) methodologies*. Tech. rep. International Council on Systems Engineering ((INCOSE)) (cit. on p. 8).

EUGSTER, P. T., P. A. FELBER, R. GUERRAOUI and A.-M. KERMARREC (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2), pp. 114–131 (cit. on p. 116).

FAN, W., J. LI, S. MA, N. TANG, Y. WU and Y. WU (2010). Graph pattern matching: from intractable to polynomial time. *Proceedings of the VLDB Endowment*, 3(1-2), pp. 264 –275 (cit. on pp. 126, 131).

FERNEY, M. (2000). Modelling and controlling product manufacturing systems using bond-graphs and state equations: Continuous systems and discrete systems which can be represented by continuous models. *Production Planning and Control*, 11(1), pp. 7–19 (cit. on pp. 3, 4).

FINLEY, J., J. PINTÉR and M. SATISH (1998). Automatic model calibration applying global optimization techniques. English. *Environmental Modeling & Assessment*, 3(1-2), pp. 117–126 (cit. on p. 176).

FLOOD, R. L. (1990). *Liberating Systems Theory*. New York: Plenum Press (cit. on pp. 12, 13).

FLOOD, R. L. and E. R. CARSON (1993). *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science*. 2nd. Springer (cit. on pp. 6, 13, 14).

FLYNN, P. J. and A. K. JAIN (1991). CAD-Based Computer Vision: From CAD Models to Relational Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2), pp. 114–132 (cit. on p. 43).

FORRESTER, J. W. (1976). *Principles of Systems*. 2nd Preliminary. Wright-Allen Press (cit. on p. 12).

FORSBERG, K. and H. MOOZ (1992). The Relationship of Systems Engineering to the Project Cycle. *Engineering Management Journal*, 4, pp. 36 –43 (cit. on p. 8).

FOWLER, J. W. and O. ROSE (2004). Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems. *Simulation*, 80(9), pp. 469–476 (cit. on pp. 2, 211).

FOWLER, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley (cit. on pp. 54, 64).

FOX, C., A. LEVITIN and T. REDMAN (1994). The notion of data and its quality dimensions. *Information Processing and Management*, 30(1), pp. 9 –19 (cit. on pp. 46, 48, 50).

FUJIMOTO, R. M. (2000). *Parallel and Distributed Simulation Systems*. Ed. by A. Y. Zomaya. Wiley Series On Parallel and Distributed Computing. John Wiley & Sons (cit. on p. 64).

FUMAROLA, M. (2011). Multiple Worlds: A multi-actor simulation-based design method for logistics systems. PhD thesis. The Netherlands: Delft University of Technology (cit. on pp. 63, 64, 120).

GALLAGHER, B. (2006). Matching structure and semantics: A survey on graph-based pattern matching. *Capturing and Using Patterns for Evidence Detection*. AAAI Press, pp. 45–53 (cit. on pp. 56, 126, 131, 132).

GAMMA, E., R. HELM, R. JOHNSON and J. VLISSIDES (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley (cit. on pp. 83, 85, 116).

GAREY, M. R. and D. S. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co. (cit. on p. 131).

GELBSTEIN, E. (2003). Data, Information, and Knowledge. *Encyclopedia of Information Systems*. Ed. by H. Bidgoli. New York: Elsevier, pp. 469 –476 (cit. on p. 46).

GELFERT, A. (2011). Mathematical formalisms in scientific practice: From denotation to model-based representation. *Studies in History and Philosophy of Science Part A*, 42(2), pp. 272 –286 (cit. on p. 25).

GELSEY, A. (1990). Automated reasoning about machines. PhD thesis. New Haven: Yale University (cit. on p. 3).

GELSEY, A. (1995). Automated reasoning about machines. *Artificial Intelligence*, 74(1), pp. 1–53 (cit. on p. 3).

GIAMBIASI, N. and J. C. CARMONA (2006). Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simulation Modelling Practice and Theory*, 14(1), pp. 47–70 (cit. on p. 88).

GIAMBIASI, N., B. ESCUDE and S. GHOSH (2000). GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. *Transactions of the SCS*, 17(3), pp. 120 –134 (cit. on pp. 69, 87, 88).

GLOTZER, S. C., S. KIM, P. T. CUMMINGS, A. DESHMUKH, M. HEAD-GORDON, G. KARNIADAKIS, L. PETZOLD, C. SAGUI and M. SHINOZUKA (2010). *International Assessment of Research and Development in Simulation-based Engineering and Science*. Tech. rep. World Technology Evaluation Center (cit. on pp. 2, 4, 211).

GOLDSMAN, D. and B. L. NELSON (1998). Comparing Systems via Simulation. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Ed. by J. Banks. Wiley Interscience. Chap. 8, pp. 273 –306 (cit. on p. 179).

GONZALEZ, R. A. (2010). A framework for ICT-supported coordination in crisis response. PhD thesis. Delft University of Technology, Faculty of Technology, Policy and Management (cit. on p. 203).

GÖSSLER, G. and J. SIFAKIS (2005). Composition for component-based modeling. *Science of Computer Programming*, 55(1-3), pp. 161–183 (cit. on pp. 23, 24).

GRANDA, J. J. and R. C. MONTGOMERY (2003). *Automated Modeling and Simulation Using the Bond Graph Method for the Aerospace Industry*. Tech. rep. CASI-20040085773. NASA Langley Research Center (cit. on pp. 2, 4).

GREGOR, S. and D. JONES (2007). The Anatomy of a Design Theory. *Journal of the Association for Information Systems*, 8(5), pp. 312–335 (cit. on p. 36).

GUBA, E. G. and Y. S. LINCOLN (1994). Competing paradigms in qualitative research. *Handbook of Qualitative Research*. Ed. by N. K. Denzin and Y. S. Lincoln. Sage Publications, pp. 105–117 (cit. on p. 6).

HABEL, A. (1992). *Hyperedge Replacement: Grammars and Languages*. Spriger-Verlag (cit. on pp. 127, 129, 133, 134, 141).

HANSEN, I. A. and J. PACHL, eds. (2008). *Railway Timetable & Traffic: Analysis-Modelling-Simulation*. Eurailpress (cit. on pp. 60, 61, 67–69, 89).

HARRISON, G. A., D. S. MAYNARD and E. POLLAK (2004). Automated database and schema-based data interchange for modeling and simulation. *Proceedings of the 2004 Winter simulation Conference*, pp. 191–197 (cit. on p. 3).

HESSE, W. (2006). More matters on (meta-)modelling: remarks on Thomas Kühne's "matters". *Software and Systems Modeling*, 5(4), pp. 387–394 (cit. on p. 22).

HEVNER, A. R., S. T. MARCH, J. PARK and S. RAM (2004). Design Science in Information Systems Research. English. *MIS Quarterly*, 28(1), pp. 75–105 (cit. on p. 8).

HO, T., B. MAO, Z. YUAN, H. LIU and Y. FUNG (2002). Computer simulation and modeling in railway applications. *Computer Physics Communications*, 143(1), pp. 1–10 (cit. on pp. 60, 61).

HO, Y.-C., ed. (1994). *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World*. IEEE (cit. on p. 27).

HOAD, K., S. ROBINSON and R. DAVIES (2010). Automated selection of the number of replications for a discrete-event simulation. *Journal of the Operational Research Society*, 61(11), pp. 1632 –1644 (cit. on pp. 177–179).

HOFMANN, M. (2004). Criteria for decomposing systems into components in modeling and simulation: Lessons learned with military simulations. *Simulation*, 80(7-8), pp. 357–365 (cit. on pp. 24, 34, 66).

HOLLANDER, Y. and R. LIU (2008). The principles of calibrating traffic microsimulation models. English. *Transportation*, 35(3), pp. 347 –362 (cit. on pp. 180, 181, 187).

HOURDAKIS, J, P. G. MICHALOPOULOS and J KOTTOMMANNIL (2003). Practical Procedure For Calibrating Microscopic Traffic Simulation Models. *Transportation Research Record: Journal of the Transportation Research Board*, (1852), pp. 130–139 (cit. on p. 181).

HUANG, Y., A. VERBRAECK, N. VAN OORT and H. VELDHOEN (2010). Rail Transit Network Design Supported by an Open Source Simulation Library: Towards Reliability Improvement. *Transportation Research Board 89th Annual Meeting Compendium of Papers*. 10-0310. Washington, DC, USA: TRB (cit. on p. 3).

JACOBS, P. H. M. (2005). The DSOL simulation suite - Enabling multi-formalism simulation in a distributed context. PhD thesis. the Netherlands: Delft University of Technology (cit. on pp. 8, 26, 63, 114, 120, 177, 183, 184, 225).

JIANG, X. Y. and H. BUNKE (1996). Including geometry in graph representations: A quadratic-time graph isomorphism algorithm and its applications. *Advances in Structural and Syntactical Pattern Recognition*. Ed. by P. Perner, P. Wang and A. Rosenfeld. Vol. 1121. LNCS. Springer Berlin Heidelberg, pp. 110 –119 (cit. on p. 146).

JIANG, X. Y. and H. BUNKE (1999). Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32(7), pp. 1273 –1283 (cit. on pp. 146, 147, 156).

JOHNSON, T., A. KERZHNER, C. PAREDIS and R. BURKHART (2012). Integrating models and simulations of continuous dynamics into SysML. *Journal of Computing and Information Science in Engineering*, 12(1) (cit. on p. 2).

JONES, L. (1982). Defining System Boundaries in Practice: Some Proposals and Guidelines. *Journal of Applied Systems Analysis*, 9, pp. 41–55 (cit. on p. 13).

JURAN, J. M. (1988). *Juran on Planning for Quality*. New York: Free Press (cit. on p. 45).

KAAS, A. (2000). Punctuality model for railways. *Advances in Transport*, 7, pp. 853–860 (cit. on p. 60).

KAHL, W. (2002). *A Relation-Algebraic Approach to Graph Structure Transformation*. Habilitationsschrift. Fakultät für Informatik, Universität der Bundeswehr München (cit. on pp. 56, 130, 132).

KAMERLING, W. (2007). Besluitvorming over traminfrastructuur. In Dutch. MA thesis. Delft Univeristy of Technology, Faculty of Technology, Policy and Management (cit. on p. 3).
KANACILO, E. M. and A. VERBRAECK (2005). A distributed multi-formalism simulation to support rail infrastructure control design. *Proceedings of the 2005 Winter Simulation Conference*. IEEE, pp. 2546–2553 (cit. on p. 114).
KANACILO, E. M. and A. VERBRAECK (2006). Simulation services to support the control design of rail infrastructures. *Proceedings of the 2006 Winter Simulation Conference*. IEEE, pp. 1372–1379 (cit. on pp. 3, 114).
KANACILO, E. M. and A. VERBRAECK (2007). Assessing tram schedules using a library of simulation components. *Proceedings of the 2007 Winter Simulation Conference*. IEEE, pp. 1878–1886 (cit. on pp. 3, 114).
KAVICKA, A. and V. KLIMA (2000). Simulation support for railway infrastructure design and planning processes. *Advances in Transport*, 7, pp. 447–456 (cit. on p. 60).
KEEN, P. and H. SOL (2008). *Decision Enhancement Services: Rehearsing the Future for Decisions that Matter*. IOS Press (cit. on p. 61).
KENNEDY, M. C. and A. O'HAGAN (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3), pp. 425 –464 (cit. on p. 176).
KÜHNE, T. (2006). Matters of (Meta-) Modeling. *Software and Systems Modeling*, 5(4), pp. 369–385 (cit. on pp. 12, 20–22).
KICKERT, W. J. M. and J. P. VAN GIGCH (1979). A Metasystem Approach to Organizational Decision-Making. *Management Science*, 25(12), pp. 1217–1231 (cit. on p. 23).
KIM, S.-J., W. KIM and L. R. RILETT (2005). Calibration of micro-simulation models using non-parametric statistical techniques. *Transportation Research Record: Journal of the Transportation Research Board*, (1935), pp. 111–119 (cit. on p. 187).
KLEIJNEN, J. P. C. (1995). Verification and validation of simulation models. *European Journal of Operational Research*, 82(1), pp. 145 –162 (cit. on pp. 179, 187).
KLEIJNEN, J. P. C. (1999). Statistical validation of simulation, including case studies. *Validation of simulation models*. Ed. by C. van Dijkum, D. DeTombe and E. van Kuijk. 112 - 125. SISWO Amsterdam (cit. on p. 179).
KLEIJNEN, J. P. C. (2009). Kriging metamodeling in simulation: A review. *European Journal of Operational Research*, 192(3), pp. 707–716 (cit. on p. 22).
KLEIJNEN, J. P. C. and R. G. SARGENT (2000). A methodology for fitting and validating metamodels in simulation. *European Journal of Operational Research*, 120(1), pp. 14–29 (cit. on p. 22).
KLEPPE, A., J. WARMER and W. BAST (2003). *MDA Explained: The Model-Driven Architecture: Practice and Promise*. Addison Wesley (cit. on p. 53).
KLIR, G. J. (2001). *Facets of Systems Science*. 2nd. Vol. 15. IFSR International Series on Systems Science and Engineering. Kluwer Academic/Plenum Publishers (cit. on pp. 12, 14, 223, 224).
KLIR, G. J. and D. ELIAS (2003). *Architecture of Systems Problem Solving*. 2nd. Vol. 21. IFSR International Series on Systems Science and Engineering. Kluwer Academic/Plenum Publishers (cit. on pp. 8, 13, 16–18, 20, 22, 24).
KLIR, G. J. (1988). The Role of Uncertainty Principles in Inductive Systems Modelling. *Kybernetes*, 17(2), pp. 24–34 (cit. on p. 18).
KNIGHT, S.-A. and J. BURN (2005). Developing a Framework for Assessing Information Quality on the World Wide Web. *Informing Science*, 8, pp. 159 –172 (cit. on p. 45).
KNUTH, D. E. (1997). *The Art of Computer Programming, Volum 1: Fundamental Algorithms*. Addison-Wesley Professional (cit. on p. 76).

KOFMAN, E. (2003a). Quantization-Based Simulation of Differential Algebraic Equation Systems. *Simulation*, 79(7), pp. 363–376 (cit. on pp. 69, 87).
KOFMAN, E. (2003b). Quantized-state control: a method for discrete event control of continuous systems. *Latin American applied research*, 33(4), pp. 339 –406 (cit. on p. 87).
KOFMAN, E., J. LEE and B. ZEIGLER (2001). DEVS Representation of Differential Equation Systems: Review of Recent Advances. *Proceedings of the 2001 European Simulation Symposium* (cit. on p. 87).
KOUTSOPOULOS, H. and Z. WANG (2007). Simulation of Urban Rail Operations: Application Framework. *Transportation Research Record: Journal of the Transportation Research Board*, 2006, pp. 84–91 (cit. on p. 60).
KREUTZER, W. (1986). *System Simulation: Programming Styles and Languages*. New York: Addison-Wesley (cit. on p. 175).
KUECHLER, B. and V. VAISHNAVI (2008). On theory development in design science research: anatomy of a research project. *European Journal of Information Systems (2008) 17, 489–504*, 17, pp. 489–504 (cit. on p. 36).
LASZLO, A. and S. KRIPPNER (1998). Systems theories: Their origins, foundations, and development. *Systems Theories and a Priori Aspects of Perception*. Ed. by J. S. Jordan. Vol. 126. Advances in Psychology. Amsterdam: Elsevier Science, pp. 47 –74 (cit. on p. 11).
LAW, A. M. (2007). *Simulation Modeling and Analysis*. 4th. McGraw-Hill (cit. on pp. 1, 14, 62, 63, 120, 122, 123, 175–179, 184, 187, 189, 195, 198, 201, 202).
LEE, J.-K., Y.-H. LIM and S.-D. CHI (2004). Hierarchical Modeling and Simulation Environment for Intelligent Transportation Systems. *Simulation*, 80(2), pp. 61–76 (cit. on p. 67).
LEE, J.-B. and K. OZBAY (2008). Calibration of a Macroscopic Traffic Simulation Model Using Enhanced Simultaneous Perturbation Stochastic Approximation Methodology. *TRB 87th Annual Meeting Compendium of Papers*, (08-2964) (cit. on p. 187).
LEE, J.-B. and K. OZBAY (2009). A New Calibration Methodology for Microscopic Traffic Simulation Using Enhanced Simultaneous Perturbation Stochastic Approximation Approach. *Transportation Research Record: Journal of the Transportation Research Board*, (2124), pp. 233–240 (cit. on p. 187).
LEE, Y. W., D. M. STRONG, B. K. KAHN and R. Y. WANG (2002). AIMQ: a methodology for information quality assessment. *Information and Management*, 40(2), pp. 133 –146 (cit. on p. 45).
LEVITIN, A. and T. REDMAN (1995). Quality dimensions of a conceptual view. *Information Processing and Management*, 31(1), pp. 81 –88 (cit. on p. 48).
LEVY, A. Y., Y. IWASAKI and R. FIKES (1997). Automated model selection for simulation based on relevance reasoning. *Artificial Intelligence*, 96(2), pp. 351–394 (cit. on p. 3).
LI, K.-P. and Z.-Y. GAO (2007). An improved equation model for the train movement. *Simulation Modelling Practice and Theory*, 15(9), pp. 1156 –1162 (cit. on p. 68).
LI, K.-P., Z.-Y. GAO and B. NING (2005). Cellular automaton model for railway traffic. *Journal of Computational Physics*, 209(1), pp. 179 –192 (cit. on p. 68).
LI, K.-P., B.-H. MAO and Z.-Y. GAO (2009). An improved walk model for train movement on railway network. *Communications in Theoretical Physics*, 51(6), pp. 979–984 (cit. on p. 68).
LIANG, V.-C. and C. PAREDIS (2004). A port ontology for conceptual design of systems. *Journal of Computing and Information Science in Engineering*, 4(3), pp. 206–217 (cit. on p. 28).
LILLRANK, P. (2003). The quality of information. *International Journal of Quality and Reliability Management*, 20(6), pp. 691 –703 (cit. on pp. 44, 45).

LITTLE, S., D. WALTER, K. JONES, C. MYERS and A. SEN (2010). Analog/mixed-signal circuit verification using models generated from simulation traces. *International Journal of Foundations of Computer Science*, 21(2), pp. 191–210 (cit. on p. 2).
LONGO, F. (2011). Advances of modeling and simulation in supply chain and industry. *Simulation*, 87(8), pp. 651–656 (cit. on pp. 2, 4, 211).
LOSHIN, D. (2011). *The Practitioner's Guide to Data Quality Improvement*. Morgan Kaufmann (cit. on pp. 44, 47, 48, 50).
LU, Q., M. DESSOUKY and R. C. LEACHMAN (2004). Modeling train movements through complex rail networks. *ACM Transactions on Modeling and Computer Simulation*, 14(1), pp. 48–75 (cit. on p. 68).
LUBARS, M. D. (1988). Wide-spectrum support for software reusability. *Software reuse: emerging technology*. Ed. by W. Tracz. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 275–281 (cit. on p. 61).
LUCKO, G., P. C. BENJAMIN, K. SWAMINATHAN and M. G. MADDEN (2010). Comparison of manual and automated simulation generation approaches and their use for construction applications. *Proceedings of the 2010 Winter Simulation Conference*, pp. 3132–3144 (cit. on p. 3).
M'PHERSON, P. K. (1974). A perspective on systems science and systems philosophy. *Futures*, 6, pp. 219–239 (cit. on p. 6).
MACK, N. K. (1990). Learning Fractions with Understanding: Building on Informal Knowledge. English. *Journal for Research in Mathematics Education*, 21(1), pp. 16–32 (cit. on p. 14).
MANIN, Y. I. (2010). *A Course in Mathematical Logic for Mathematicians*. Vol. 53. Graduate Texts in Mathematics. Springer New York (cit. on p. 127).
MANNING, C. D., P. RAGHAVAN and H. SCHÜTZE (2008). *Introduction to Information Retrieval*. Cambridge University Press (cit. on p. 157).
MARCH, S. T. and G. F. SMITH (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4), pp. 251 –266 (cit. on p. 8).
MASON, D. M. and J. H. SCHUENEMEYER (1983). A Modified Kolmogorov-Smirnov Test Sensitive to Tail Alternatives. *The Annals of Statistics*, 11(3), pp. 933 –946 (cit. on p. 195).
MCGILVRAY, D. (2008). *Executing Data Quality Projects: Ten Steps to Quality Data and Trusted Information*. Morgan Kaufmann (cit. on pp. 48, 50).
MENS, T. and P. VAN GORP (2006). A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152, pp. 125–142 (cit. on pp. 40, 53–55).
MIDDELKOOP, D. and M. BOUWMAN (2001). Simone: Large Scale Train Network Simulations. *Proceedings of the 2001 Winter Simulation Conference*. IEEE, pp. 1042–1047 (cit. on pp. 60, 61, 68).
MIELCZAREK, B. and J. UZIALKO-MYDLIKOWSKA (2012). Application of computer simulation modeling in the health care sector: a survey. *Simulation*, 88(2), pp. 197–216 (cit. on p. 2).
MINGERS, J. (2006). *Realising Systems Thinking: Knowledge and Action in Management Science*. Ed. by R. L. Flood. Vol. 14. Contemporary systems thinking. Springer US (cit. on p. 6).
MORRIS, C. W. (1938). *Foundations of the Theory of Signs*. Vol. 1. International Encyclopaedia of Unified Sciences. University of Chicago Press (cit. on p. 45).
MORRIS, C. W. (1971). *Writings on the General Theory of Signs*. Vol. 1. International Encyclopaedia of Unified Sciences. The Hague & Paris: Mouton (cit. on p. 45).
MOSER, H. G. (2006). Component-based Modelling and Simulation. MA thesis. Royal Institute of Technology (KTH), Sweden and Nanyang Technological University (NTU), Singapore (cit. on p. 24).

MUELLER, R. (2007). Specification And Automatic Generation Of Simulation Models With Applications In Semiconductor Manufacturing. PhD thesis. Georgia Institute of Technology (cit. on pp. 3, 4).
MUSSELMAN, K. J. (1998). Guidelines for Success. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Ed. by J. Banks. Wiley Interscience. Chap. 22, pp. 721–743 (cit. on pp. 57, 176).
NASH, A. and D. HUERLIMANN (2004). Railroad simulation using OpenTrack. *Advances in Transport*, 15, pp. 45–54 (cit. on p. 60).
NAYAK, P. (1995). *Automated Modeling of Physical Systems*. Vol. 1003. Lecture Notes in Computer Science (cit. on p. 2).
NIERSTRASZ, O. and T. D. MEIJLER (June 1995). Research directions in software composition. *ACM Comput. Surv.* 27(2), pp. 262–264 (cit. on p. 25).
NIERSTRASZ, O. and D. TSICHRITZIS, eds. (1995). *Object-Oriented Software Composition*. Object-Oriented Series. Prentice Hall PTR (cit. on p. 25).
NÖTH, W. (1995). *Handbook of Semiotics*. Indiana University Press (cit. on p. 45).
OMG (2003). *MDA Guide*. Version 1.0.1. Object Management Group (cit. on p. 22).
ÖREN, T. I. (1981). Concepts and criteria to assess acceptability of simulation studies: a frame of reference. *Communications of The ACM*, 24(4), pp. 180 –189 (cit. on p. 62).
ÖREN, T. I. and B. P. ZEIGLER (2012). System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore. *Simulation*, 88(9), pp. 1033–1046 (cit. on pp. 11, 13, 32).
ORTÚZAR, J. and L. WILLUMSEN (2001). *Modelling Transport*. 3rd. John Wiley & Sons (cit. on p. 60).
OVERSTREET, C. M. (1982). Model Specification and Analysis for Discrete-Event Simulation. PhD thesis. Virginia Tech, Department of Computer Science (cit. on p. 12).
OVERTON, D. (1989). Traffic signal control of LRVs. *IEE Colloquium on Light Rapid Transit On-Street*, pp. 9/1–9/3 (cit. on p. 60).
PACHL, J. (2002). *Railway Operation and Control*. VTD Rail Publishing (cit. on pp. 60, 67, 70, 102, 136).
PAGE, E. H., B. R. and J. A. TUFAROLO (2004). Toward a family of maturity models for the simulation interconnection problem. *Proceedings of the Spring Simulation Interoperability Workshop*. IEEE CS Press (cit. on p. 24).
PAGE, E. H. (1995). Simulation modeling methodology: principles and etiology of decision support. PhD thesis. Virginia Tech (cit. on pp. 2, 12).
PAGE, E. H. (2007). Theory and Practice for Simulation Interconnection: Interoperability and Composability in Defense Simulation. *CRC Handbook of Dynamic Systems Modeling*. Ed. by P. A. Fishwick. CRC Press. Chap. 16 (cit. on p. 24).
PALACZ, W. (2004). Algebraic hierarchical graph transformation. *Journal of Computer and System Sciences*, 68(3), pp. 497 –520 (cit. on p. 127).
PAREDIS, C., A. DIAZ-CALDERON, R. SINHA and P. KHOSLA (2001). Composable models for simulation-based design. *Engineering with Computers*, 17(2), pp. 112–128 (cit. on p. 28).
PARNAS, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of ACM*, 15(12), pp. 1053 –1058 (cit. on pp. 64, 66, 67).
PETTER, S., W. DELONE and E. MCLEAN (2008). Measuring information systems success: models, dimensions, measures, and interrelationships. *European Journal of Information Systems*, 17, pp. 236 –263 (cit. on p. 45).

PETTY, M. D. and E. W. WEISEL (2003a). A composability lexico. *Proceedings of the Spring 2003 Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization, pp. 181–187 (cit. on pp. 24, 212).
PETTY, M. D. and E. W. WEISEL (2003b). A formal basis for a theory of semantic composability. *Proceedings of the Spring 2003 Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization (cit. on pp. 24, 25).
PIDD, M. and A. CARVALHO (2006). Simulation software: not the same yesterday, today or forever. *Journal of Simulation*, 1(1), pp. 7 –20 (cit. on p. 62).
PIPINO, L. L., Y. W. LEE and R. Y. WANG (2002). Data quality assessment. *Communications of the ACM*, 45(4), pp. 211 –218 (cit. on pp. 44, 45, 47, 48, 50).
PRICE, R. and G. SHANKS (2005). A semiotic information quality framework: development and comparative analysis. *Journal of Information Technology*, 20, pp. 88 –102 (cit. on pp. 44–50).
PROCTOR, S (1998). Linking philosophy and method in the research process: the case for realism. *Nurse Researcher*, 5(4), pp. 73–90 (cit. on p. 6).
RAATIKAINEN, K. E. E. (1993). A Sequential Procedure for Simultaneous Estimation of Several Means. *ACM Transactions on Modeling and Computer Simulation*, 3, pp. 108–133 (cit. on p. 179).
REDMAN, T. (1996). *Data Quality for the Information Age*. Artech House (cit. on pp. 46, 47, 50).
REKAPALLI, P. V. (2008). Discrete-event simulation based virtual reality environments for construction operations. PhD thesis. Purdue University, Civil Engineering (cit. on p. 123).
RIZZOLI, A. E., N. FORNARA and L. M. GAMBARDELLA (2002). A simulation tool for combined rail/road transport in intermodal terminals. *Journal of Mathematics and Computers in Simulation*, 59(1-3), pp. 57–71 (cit. on p. 60).
ROBINSON, S. (2004). *Simulation: The Practice of Model Development and Use*. John Wiley & Sons (cit. on pp. 62, 65, 177).
ROBINSON, S. (2010). Conceptual Modeling for Simulation: Definition and Requirements. *Conceptual Modeling for Discrete-Event Simulation*. Ed. by S. Robinson, R. Brooks, K. Kotiadis and D.-J. V. D. Zee. CRC Press. Chap. 1, pp. 3–30 (cit. on p. 65).
ROBINSON, S., R. E. NANCE, R. J. PAUL, M. PIDD and S. J. TAYLOR (2004). Simulation model reuse: definitions, benefits and obstacles. *Simulation Modelling Practice and Theory*, 12(7–8). Simulation in Operational Research, pp. 479–494 (cit. on p. 61).
RÖHL, M. and S. MORGENSTERN (2007). Composing simulation models using interface definitions based on web service descriptions. *Proceedings of the 2007 Winter Simulation Conference*, pp. 815–822 (cit. on p. 28).
ROMAN, M. and D. SELISTEANU (2012). Pseudo bond graph modeling of wastewater treatment bioprocesses. *Simulation*, 88(2). cited By (since 1996) 1, pp. 233–251 (cit. on pp. 3, 4).
ROWLEY, J. E. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2), pp. 163–180 (cit. on p. 44).
ROYCE, W. W. (1970). Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, pp. 1–9 (cit. on p. 8).
ROYCHOUDHURY, I., M. DAIGLE, G. BISWAS and X. KOUTSOUKOS (2011). Efficient simulation of hybrid systems: A hybrid bond graph approach. *Simulation*, 87(6), pp. 467–498 (cit. on p. 2).
RUDOLPH, R. (2000). Operational simulation of light rail systems. *Proceedings of the European Transport Conference*. 167-178 (cit. on pp. 60, 61).
SAGE, A. P. (1992). *Systems Engineering*. Wiley IEEE (cit. on p. 8).

SANDBLAD, B., A. ANDERSSON, K.-E. JONSSON, P. HELLSTRÖM, P. LINDSTRÖM, J. RUDOLF, J. STORCK and M. WAHL-BORG (2000). A train traffic operation and planning Simulator. *Advances in Transport*, 7, pp. 241–248 (cit. on p. 60).

SARGENT, R. G. (2001). Some approaches and paradigms for verifying and validating simulation models. *Proceedings of the 2001 Winter Simulation Conference*, pp. 106–113 (cit. on p. 176).

SARGENT, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation*, 7(1). cited By (since 1996)0, pp. 12 –24 (cit. on p. 179).

SARJOUGHIAN, H. S. (2006). Model composability. *Proceedings of the 2006 Winter simulation Conference*. Monterey, California, pp. 149–158 (cit. on p. 24).

SCANNAPIECO, M., P. MISSIER and C. BATINI (2005). Data Quality at a Glance. *Datenbank-Spektrum*, 14 (cit. on pp. 46, 50).

SCHUNN, C. D. and D. WALLACH (2005). Evaluating goodness-of-fit in comparison of models to data. *Psychologie der Kognition: Reden and Vorträge anlässlich der Emeritierung von Werner Tack*. Ed. by W. Tack. Saarbrueken, Germany: University of Saarland Press (cit. on pp. 180, 198).

SECK, M. D. and A. VERBRAECK (2009). DEVS in DSOL: Adding DEVS operational semantics to a generic Event-Scheduling Simulation Environment. *Proceedings of the 2009 Summer Computer Simulation Conference* (cit. on pp. 63, 114, 225).

SHANKS, G. and B. CORBITT (1999). Understanding Data Quality: Social and Cultural Aspects. *Proceeding of the 10th Australasian Conference on Information Systems*, pp. 785 –797 (cit. on p. 49).

SHANNON, R. E. (1975). *Systems simulation: the art and science*. Prentice Hall, Inc. (cit. on pp. 1, 2, 12, 15, 63–65, 223).

SHEATHER, S. J. and M. C. JONES (1991). A Reliable Data-Based Bandwidth Selection Method for Kernel Density Estimation. English. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(3), pp. 683 –690 (cit. on p. 189).

SIDI, F., P. SHARIAT PANAHY, L. AFFENDEY, M. JABAR, H. IBRAHIM and A. MUSTAPHA (2012). Data quality: A survey of data quality dimensions. *Proceedings of 2012 International Conference on Information Retrieval and Knowledge Management*, pp. 300 –304 (cit. on p. 45).

SIMON, H. A. (1962). The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106(6), pp. 467–482 (cit. on pp. 13, 14).

SIMON, H. A. (1996). *The Sciences of the Artificial*. 3rd. MIT Press (cit. on pp. 8, 13, 14, 24, 66, 99).

SJOBERG, D. I. K., T. DYBA and M. JORGENSEN (2007). The Future of Empirical Methods in Software Engineering Research. *2007 Future of Software Engineering*. FOSE 2007. Washington, DC, USA: IEEE Computer Society, pp. 358–378 (cit. on p. 9).

SKIENA, S. S. (1997). *The Algorithm Design Manual*. Springer-Verlag New York (cit. on pp. 131, 132).

SOL, H. G. (1982). Simulation in Information Systems Development. PhD thesis. The Netherlands: University of Groningen (cit. on pp. 1, 61).

SOMMERVILLE, I. (1996). *Software Engineering*. 5th. Addison-Wesley (cit. on pp. 59, 64).

SOROOSHIAN, S., V. K. GUPTA and J. L. FULTON (1983). Evaluation of Maximum Likelihood Parameter estimation techniques for conceptual rainfall-runoff models: Influence of calibration data variability and length on model credibility. *Water Resources Research*, 19(1), pp. 251 –259 (cit. on pp. 176, 180, 181).

SPRINKLE, J., B. RUMPE, H. VANGHELUWE and G. KARSAI (2011). Metamodelling: State of the art and research challenges. *Model-Based Engineering of Embedded Real-Time Systems*. Ed. by

H. Giese, G. Karsai, E. Lee, B. Rumpe and B. Schätz. Vol. 6100. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, pp. 57–76 (cit. on pp. 21, 22).
STACHOWIAK, H (1973). *Allgemeine Modelltheorie*. In German. Vienna: Springer (cit. on p. 12).
SYRIANI, E. (2011). A Multi-Paradigm Foundation for Model Transformation Language Engineering. PhD thesis. McGill University, School of Computer Science (cit. on pp. 53, 56).
SZABO, C. and Y. M. TEO (2007). On Syntactic Composability and Model Reuse. *Proceedings of the International Conference on Modeling and Simulation*. IEEE Computer Society Press, pp. 230–237 (cit. on pp. 24, 64, 212).
SZYPERSKI, C. (2011). *Component Software: Beyond Object-oriented Programming*. 2nd. Addison-Wesley component software series. Addison-Wesley (cit. on pp. 24, 64, 212).
TAHMASSSEBY, S. (2009). Reliability in Urban Public Transport Network Assessment and Design. PhD thesis. The Netherlands: Delft University of Technology (cit. on pp. 185, 195).
TALPAZ, H., G. DA ROZA and A. HEARN (1987). Parameter estimation and calibration of simulation models as a non-linear optimization problem. *Agricultural Systems*, 23(2), pp. 107 –116 (cit. on p. 176).
TAYLOR, J. R. (1999). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. 2nd. University Science Books (cit. on p. 48).
TEJAY, G., G. DHILLON and A. CHIN (2006). Data quality dimensions for information systems security: A theoretical exposition. *IFIP International Federation for Information Processing*, 193 (cit. on p. 45).
THEEG, G. and S. VLASENKO, eds. (2009). *Railway Signalling & Interlocking: International Compendium*. Eurailpress (cit. on pp. 67, 68, 70, 102, 103, 105).
THOMASETH, K. (2003). Multidisciplinary modelling of biomedical systems. *Computer Methods and Programs in Biomedicine*, 71(3), pp. 189–201 (cit. on p. 3).
TIAN, Y., B. LIU, H.-W. GAO and W.-Q. LI (2012). Modeling and simulation of electro-hydraulic proportional position control system with the flexible hose. *Advanced Materials Research*, 468-471, pp. 2094–2099 (cit. on p. 2).
TOLK, A., L. BAIR and S. DIALLO (2013a). Supporting network enabled capability by extending the levels of conceptual interoperability model to an interoperability maturity model. *Journal of Defense Modeling and Simulation*, 10(2), pp. 145–160 (cit. on p. 24).
TOLK, A., ed. (2013). *Ontology, Epistemology, and Teleology for Modeling and Simulation: Philosophical Foundations for Intelligent M&S Applications*. Intelligent Systems Reference Library. Springer Berlin Heidelberg (cit. on p. 6).
TOLK, A., S. Y. DIALLO, R. D. KING, C. D. TURNITSA and J. J. PADILLA (2010). Conceptual Modeling for Composition of Model-based Complex Systems. *Conceptual Modeling for Discrete-Event Simulation*. Ed. by S. Robinson, R. Brooks, K. Kotiadis and D.-J. V. D. Zee. CRC Press. Chap. 14, pp. 355–381 (cit. on pp. 24, 64, 66, 212).
TOLK, A., S. Y. DIALLO, J. J. PADILLA and H. HERENCIA-ZAPANA (2013b). Reference modelling in support of M&S - foundations and applications. *Journal of Simulation*, 7(2), pp. 69 –82 (cit. on p. 11).
UHRMACHER, A. M. (2001). Dynamic structures in modeling and simulation: a reflective approach. *ACM Transactions on Modeling and Computer Simulation*, 11(2), pp. 206 –232 (cit. on pp. 112, 225).
ULGAN, O. and A. GUNAL (1998). Simulation in the Automobile Industy. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Ed. by J. Banks. Wiley Interscience, pp. 547–570 (cit. on p. 61).

ULLMANN, J. R. (1976). An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1), pp. 31 –42 (cit. on p. 131).
ULRICH, W. (2001). A Philosophical Staircase for Information Systems Definition, Design and Development: A Discursive Approach to Reflective Practice in ISD (Part 1). *Journal of Information Technology Theory and Application (JITTA)*, 3(3), pp. 55–84 (cit. on p. 44).
UMESH RAI, B. and L. UMANAND (2009). Bond graph toolbox for handling complex variable. *IET Control Theory and Applications*, 3(5), pp. 551–560 (cit. on p. 2).
VALENTIN, E. (2011). Effective simulation studies using domain specific simulation building blocks. PhD thesis. Delft University of Technology, Faculty of Technology, Policy and Management (cit. on p. 24).
VAN DER PIJL, G. J. (1994). Measuring the strategic dimensions of the quality of information. *The Journal of Strategic Information Systems*, 3(3), pp. 179 –190 (cit. on p. 45).
VAN ANTWERPEN, H. (2011). *Rail Simulation Suite Development*. BA Thesis, De Haagse Hogeschool (cit. on p. 219).
VAN GIGCH, J. P. (1991). *System Design Modeling and Metamodeling*. Plenum Press (cit. on p. 23).
VAN GIGCH, J. P. (1993). Metamodeling: The epistemology of system science. *Systems Practice*, 6(3), pp. 251–258 (cit. on p. 23).
VAN GIGCH, J. P. (2005). Metadecisions: Invoking the epistemological imperative to enhance the meaning of knowledge for problem solving. *Systems Research and Behavioral Science*, 22(1), pp. 83–89 (cit. on p. 23).
VAN OORT, N. (2011). Service Reliability and Urban Public Transport Design. PhD thesis. Netherlands: Delft University of Technology, Department of Transport and Planning (cit. on pp. 3, 185, 195).
VANGHELUWE, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. *IEEE International Symposium on Computer-Aided Control System Design*, pp. 129 –134 (cit. on pp. 26, 29).
VANGHELUWE, H. (2008). Foundations of Modelling and Simulation of Complex Systems. *Electronic Communications of the EASST - Proceedings of the 7th International Workshop on Graph Transformation and Visual Modeling Techniques*. 10 (cit. on pp. 13, 29).
VANGHELUWE, H. and J. DE LARA (2002). Meta-Models are Models too. *Proceedings of the 2002 Winter Simulation Conference*, pp. 597–605 (cit. on p. 20).
VANNI, T., J. KARNON, J. MADAN, R. WHITE, W. EDMUNDS, A. FOSS and R. LEGOOD (2011). Calibrating models in economic evaluation: a seven-step approach. English. *PharmacoEconomics*, 29(1), pp. 35 –49 (cit. on pp. 179, 180).
VARRÓ, D. and A. BALOGH (2007). The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, 68(3). Special Issue on Model Transformation, pp. 214 – 234 (cit. on p. 56).
VARRÓ, G., A. HORVÁTH and D. VARRÓ (2008). Recursive Graph Pattern Matching. *Applications of Graph Transformations with Industrial Relevance*. Ed. by A. Schürr, M. Nagl and A. Zündorf. Berlin, Heidelberg: Springer-Verlag, pp. 456–470 (cit. on pp. 56, 154).
VELDHOEN, H. (2009). Embedding Simulation in Decision Making. MA thesis. Delft University of Technology, Faculty of Technology, Policy and Management (cit. on pp. 3, 185).
VELEGRAKIS, Y., J. MILLER and L. POPA (2004). Preserving mapping consistency under schema changes. *The VLDB Journal*, 13(3), pp. 274 –293 (cit. on p. 51).
VERBRAECK, A., Y. SAANEN, Z. STOJANOVIC, E. VALENTIN, K. VAN DER MEER and A. M. B. SHISHKOV (2002). Building blocks for Effective Telematics Application Development and Evaluation. Ed.

by A. Verbraeck and A. Dahanayak. TU Delft. Chap. What are building blocks?, pp. 8–21 (cit. on pp. 23, 24).

VIEGA, J., B. TUTT and R. BEHRENDS (1998). *Automated Delegation is a Viable Alternative to Multiple Inheritance in Class Based Languages*. Tech. rep. Charlottesville, VA, USA: Cs-98-03, Microsoft Corporation (cit. on pp. 83, 85).

VILLA, F., A. VOINOV, C. FITZ and R. COSTANZA (2004). Calibration of Large Spatial Models: A Multistage, Multiobjective Optimization Technique. *Landscape Simulation Modeling*. Ed. by R. Costanza and A. Voinov. Modeling Dynamic Systems. Springer New York, pp. 77–116 (cit. on p. 181).

VON BERTALANFFY, L. (1950). An Outline of General System Theory. *The British Journal for the Philosophy of Science*, 1(2), pp. 134 –165 (cit. on p. 11).

VROMANS, M. J. C. M., R. DEKKER and L. G. KROON (2006). Reliability and heterogeneity of railway services. *European Journal of Operational Research*, 172(2), pp. 647–665 (cit. on p. 61).

VUCHIC, V. R. (2005). *Urban Transit: Operations, Planning, and Economics*. John Wiley & Sons, Inc (cit. on pp. 67, 136).

WAHLBORG, M. (1996). Simulation models: Important aids for Banverket's planning process. *Computers in Railways*. Vol. V. 1. WIT Press, pp. 175–181 (cit. on p. 60).

WAINER, G. A. (2009). *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press (cit. on pp. 28, 29, 64, 87, 88, 112, 120).

WAINER, G. (2006). ATLAS: A language to specify traffic models using Cell-DEVS. *Simulation Modelling Practice and Theory*, 14(3), pp. 313–337 (cit. on p. 67).

WALICKI, M., A. HODZIC and S. MELDAL (2001). Compositional Homomorphisms of Relational Structures. *Fundamentals of Computation Theory*. Ed. by R. Freivalds. Vol. 2138. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, pp. 359–371 (cit. on p. 224).

WAND, Y. and R. Y. WANG (1996). Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11), pp. 86 –95 (cit. on pp. 44, 46, 48, 49).

WANG, J., Q. CHANG, G. XIAO, N. WANG and S. LI (2011). Data driven production modeling and simulation of complex automobile general assembly plant. *Computers in Industry*, 62(7), pp. 765–775 (cit. on pp. 3, 4).

WANG, R. and D. STRONG (1996). Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4), pp. 5 –34 (cit. on pp. 45, 47, 48, 50).

WANG, Y. (2009). Toward a formal knowledge system theory and its cognitive informatics foundations. *Transactions on Computer Science*, 5540 LNCS, pp. 1–19 (cit. on p. 15).

WARNER, S. (1990). *Modern Algebra*. Courier Dover Publications (cit. on p. 224).

WASHIO, T. and H. MOTODA (2003). State of the art of graph-based data mining. *ACM SIGKDD Explorations Newsletter*, 5(1), pp. 59 –68 (cit. on p. 132).

WASYNCZUK, O. and S. SUDHOFF (1996). Automated state model generation algorithm for power circuits and systems. *IEEE Transactions on Power Systems*, 11(4), pp. 1951–1956 (cit. on p. 2).

WEBER, R. (2004). The Rhetoric of Positivism Versus Interpretivism: A Personal View. *MIS Quarterly*, 28(1), pp. iii–xii (cit. on pp. 6, 7).

WEISSTEIN, E. W. (2009). *Graph Automorphism*. MathWorld - A Wolfram Web Resource (cit. on p. 155).

WETZEL, L. (2011). Types and Tokens. *The Stanford Encyclopedia of Philosophy*. Ed. by E. N. Zalta (cit. on p. 20).

WIELAND, F. and A. PRITCHETT (2007). Looking into the Future of Air Transportation Modeling and Simulation: A Grand Challenge. *Simulation*, 83(5), pp. 373–384 (cit. on pp. 2, 4, 211).

WYMORE, A. W. (1967). *A mathematical theory of systems engineering: the elements*. Wiley (cit. on pp. 8, 11, 12, 32).
XIA, S. and N. SMITH (1996). Automated modelling: A discussion and review. *Knowledge Engineering Review*, 11(2), pp. 137–160 (cit. on p. 2).
YAPO, P. O., H. V. GUPTA and S. SOROOSHIAN (1996). Automatic calibration of conceptual rainfall-runoff models: sensitivity to calibration data. *Journal of Hydrology*, 181(1–4), pp. 23 –48 (cit. on pp. 176, 180, 181).
YILMAZ, L. (2004). On the Need for Contextualized Introspective Models to Improve Reuse and Composability of Defense Simulations. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1(3), pp. 141–151 (cit. on pp. 24, 64, 212).
ZAGLAUER, S. and U. KNOLL (2012). Evolutionary algorithms for the automatic calibration of simulation models for the virtual engine application. *7th Vienna International Conference on Mathematical Modelling*. Ed. by I. Troch and F. Breitenecker. Vol. 7. 1, pp. 177–181 (cit. on p. 176).
ZANGWILL, O. L. (1980). Kenneth Craik: the man and his work. *The British journal of psychology*, 71(1), pp. 1–16 (cit. on p. 12).
ZEIGLER, B. P. (1976). *Theory of Modeling and Simulation*. 1st. New York: Wiley Interscience (cit. on p. 27).
ZEIGLER, B. P. (2003). Systems Movement: Autobiographical Retrospectives – The Evolution of Theory of Modeling and Simulation. *International Journal of General Systems*, 32(3), pp. 221–236 (cit. on pp. 8, 11, 13).
ZEIGLER, B. P. and J. S. LEE (1998). Theory of quantized systems: Formal basis for DEVS/HLA distributed simulation environment. *Proceedings of SPIE - The International Society for Optical Engineering*. Vol. 3369, pp. 49–58 (cit. on p. 69).
ZEIGLER, B. P., H. PRAEHOFER and T. G KIM (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2nd. Elsevier/Academic Press (cit. on pp. 13, 14, 16, 18, 19, 24–30, 34, 62–65, 175, 176, 223).
ZEIGLER, B. P. (1998). *DEVS Theory of Quantized Systems*. Advance Simulation Technology Thrust (ASTT), DARPA Contract N6133997K-0007. University of Arizona (cit. on pp. 29, 87).
ZHANG, X., G. HÖRMANN, N. FOHRER and J. GAO (2012). Parameter calibration and uncertainty estimation of a simple rainfall-runoff model in two case studies. *Journal of Hydroinformatics*, 14(4), 1061 – 1074 (cit. on p. 176).
ZINS, C. (2007). Conceptual approaches for defining data, information, and knowledge. *Journal of the American Society for Information Science and Technology*, 58(4), pp. 479–493 (cit. on p. 44).
ZUPANCIC, B. and A. SODJA (2012). Computer-aided physical multi-domain modelling: Some experiences from education and industrial applications. *Simulation Modelling Practice and Theory*. Article in Press (cit. on p. 2).

List of Tables

List of Figures

Subject Index

Summary

Automated Simulation Model Generation

One challenge in today's Modeling and Simulation (M&S) projects is to model increasingly larger and more complex systems. Complex simulation models take long to develop and incur high costs. While there is a rich history of efforts to improve modeling processes, there is still considerable room for improvement. With the advances in data collection technology and more popular use of computer-aided systems, more and more data has become available in many organizations. The increased availability of data, on the one hand, could foster automation in modeling, and on the other hand often requires a certain degree of automation since the large amount of data can no longer be manually handled effectively and efficiently.

In this research, we studied Automated Model Generation (AMG). The objective was *to provide a method that automatically generates simulation models with flexible structures using existing data assuming that these simulation models are intended for a certain domain.* The AMG method provided by this thesis differs from other AMG research in at least two aspects. First, the data used for the AMG does not contain specifications of the model structure. Second, the generated models have structures that are dynamically constructed during the AMG. We decomposed the research objective into four interrelated tangible parts and identified the following research questions:

1. *What is a good way to define flexible structures for simulation models in order to achieve the research objective?*
2. *What are the requirements for the data in order to achieve the research objective?*
3. *What functionalities should a method provide in order to automatically generate simulation models with flexible structures using existing data?*
4. *What is the quality of the simulation model generated by the method?*

We used cases in the domain of light-rail transport to study the AMG method. Since the method aims at generating simulation models intended for a certain domain, a good way to define flexible model structures is required to represent a class of models of interest as a prerequisite for the AMG. For this purpose, we propose domain metamodels and domain model components that should represent different knowledge levels of the systems of interest, implement the concept of component-based modeling, and

use an appropriate underlying modeling formalism. Domain meta-models allow for an abstract representation of model structures of a class of models, and domain model components specify concrete model behaviors that are used as building blocks for model composition.

We developed a domain model component library, LIBROS (Library for Rail Operations Simulation) using DEVS (Discrete Event System Specification), whose design complies with the above concepts. The atomic models in LIBROS specify model behaviors at the elementary level. The coupled models are defined to represent domain meta-models that allow for a set of model compositions. Although the statement of "components should support composition" may seem redundant, many researchers acknowledge that model composability is difficult to apply. To support a good level of modularity and composability, we designed cohesive model ports and coupling relations and a communication mechanism. Based on message propagation, the communication mechanism is suitable for decentralized communications in a connected network model. A number of other modeling practices were also discussed: a SAM (shared alterable mixin) design pattern for atomic model behavior composition, reuse and extension; the handling of interrupts in Generalized DEVS (GDEVS); vehicle model couplings with Dynamic Structure DEVS (DSDEVS); a State Update and Transition Mechanism (SUTM), its benefits and potential computational costs.

For the AMG method, existing data is supposed to provide concrete information about model structures and parameterization. Typically, existing data has quality issues and does not contain all types of information, particularly in terms of model structures, that are required for the AMG. In designing the AMG method, the information gap should be identified and measures are defined accordingly to close this gap. The functionalities that should be provided by the AMG method basically include those measures that are automatable. In the generation of light-rail simulation models studied by this research, infrastructure CAD data is used as a basis for model structures. Certain measures are taken to solve data issues such as mapping consistency and pragmatic completeness. For example, minor modifications, such as the locations where the vehicle models shall be generated in the simulation, are made manually to the data to solve the issue of pragmatic completeness for the AMG. The major challenge for the AMG method is, however, the presentation suitability in model structures (including identification of model components and model compositional relations). Presentation suitability refers to the degree to which the data is appropriate for the purpose of data use in terms of format, unit, precision and type-sufficiency, where type-sufficiency refers to the degree to which the data includes all the types of useful information. To solve this problem, the types of information that are available in the data and those that can be inferred from the data (which are required for the AMG) are arranged by their dependencies. Based on these dependencies, three transformation steps (with sub-steps) are designed to infer the required information from the data step by step and to finally generate simulation models.

The data structure representations and transformations in the AMG method are based on graph transformation theory. A pair of an ordinary digraph and a hypergraph is used to represent a model composite graph (MCG) which has a compositional con-

tainment hierarchy (CCH). A CCH is a strictly nested component inclusion hierarchy. A set of transformation rules is specified on the meta-models of the original data structure, of intermediate structures and of the simulation model as a final outcome. The meta-models of the data and the data structure compositions of interest (for transformation) are defined as graph patterns and pattern composites whose matching instances are recorded as hyperedges in the MCG.

In the first transformation step, a digraph that represents the rail infrastructure network (as a flat graph) is constructed from the original infrastructure data (which does not contain vertex relations) based on geometrical inferences. In the second transformation step, graph pattern matching and hyperedge replacement are performed incrementally (with partially ordered sub-steps where composition and decomposition are both used) on the graphs which produce an MCG. For the transformation, we defined a number of graph patterns and pattern composites, an algorithm for ordered composite isomorphism for the matching of the defined graph patterns, and an algorithm for the merging of composites within a defined distance measure. The purpose of the MCG is to aggregate the vertices in the digraph into a CCH such that the matched graph patterns can be transformed into corresponding model components in the next step. In the third transformation step, the vertices and hyperedges in the MCG are transformed into atomic and coupled model components corresponding to the vertex and hyperedge types, and the vertex relations are transformed into model coupling relations. In this context, a hyperedge (composite) is injective homomorphic to the (generated) coupled infrastructure model, where a number of sub-components are added into the model according to the meta-model definitions. Among these sub-components, the control logic of each crossing model component is generated according to the corresponding infrastructure layout. After these steps, we obtain a simulation model where the model behavior at the elementary level is pre-specified in the atomic components in the domain model component library, and the model structure is dynamically constructed using the coupled components according to the infrastructure data.

To complement the AMG method, when operational data from the real system is available, the generated model can be calibrated by a model calibration procedure using user defined goodness-of-fit measures and parameter search algorithms. In the calibration test presented in this thesis, a preliminary study was made to calibrate interval driving times of the generated LIBROS light-rail simulation model of The Hague, and the issues related to the calibration experiments are discussed. Note that the goodness-of-fit measures in the calibration procedure can also serve as a way to validate the relevant model output data (operational validation through comparison).

During the research and model development process, we closely followed guidelines discussed in M&S literature in order to build valid and creditable models. As a final step of model validation in this research, and a way to evaluate the AMG method, a validation session (with a combination of questionnaires and discussions) was organized with a panel of nine subject-matter experts. Specific questions are designed to ask about the model behaviors and interactions of the generated LIBROS light-rail simulation models of The Hague in order to evaluate the structural validity of the models. As follow-ups for the validation session, minor adjustments in the model and paramet-

ers adjustments for the model generation were made to fine-tune the AMG outcomes. The experts raised no substantial modeling problems in the validation session. Overall, they appreciated the fact that complex simulation models can be generated using infrastructure data, and showed support for the structural validity of the model for simulation studies such as timetable and driving time analysis. They also suggested further directions for model development.

In this research, through the cases in the domain of light-rail transport, we designed a method that can automatically generate valid simulation models with flexible structures using existing data and a domain model component library. The library was designed to support a good level of modularity and composability. It contains domain model components that specify concrete model behaviors (at the elementary level) which are used as AMG building blocks, as well as components that represent domain meta-models which allow for a set of model compositions to be dynamically constructed during the AMG. In order to close the information gap between the existing data and the information that is required for model structures, a set of transformation rules was specified. For the transformations, graph patterns and pattern composites were defined, and graph pattern matching and hyperedge replacement algorithms were performed incrementally to produce a model composite graph, according to which a simulation model was generated.

In general, a good design of flexible model structures is the outcome of joint considerations: a generic modeling formalism, formal definitions of model structures, separation of concerns and reusable definitions of model behaviors, cohesive definitions of model interfaces and interaction, etc. The transformation rules for the AMG can be defined when the data as AMG input has semantic and pragmatic completeness, has definable measures for syntactic and mapping inconsistency (if any), and when the modelers have sufficient domain knowledge and deductive reasoning for the definition of transformation rules that can solve data issues related to semantic accuracy and presentation suitability with regard to model structure and parameterization. The rules are defined on the meta-models of the original data structure, of intermediate structures, and of the simulation model in order to construct a representation of the model structure whose components can be mapped to the corresponding pre-defined simulation model components, and to construct a simulation model according to the representation of the model structure.

Founded on rich systems theory, M&S theory, graph and hypergraph transformation theory, and accompanied by practical applications of the method (and the generated models) in the domain of light-rail transport, we argue that the method has a strong potential for applications to graph-representable systems such as those in the infrastructure domain. Future research includes studies for calibration methods and techniques, extension of the LIBROS simulation library, multi-resolution and multi-perspective model generation, and applications of the AMG method to other domains.

Samenvatting

Automatische Generatie van Simulatiemodellen

Een van de uitdagingen voor simulatieprojecten is het modelleren van steeds grotere en complexere systemen. Het ontwikkelen van complexe simulatiemodellen duurt echter lang en kost veel. Ondanks het feit dat er de afgelopen tijd veel vooruitgang is geboekt met de kwaliteit van het modelleerproces, zijn er nog veel mogelijkheden voor verdere verbetering. Door ontwikkelingen in dataverzamelingsmethoden en de populariteit van CAD-systemen, beschikken steeds meer bedrijven over grote hoeveelheden digitale data. Aan de ene kant stelt de grotere beschikbaarheid van data ons in staat om meer automatisering toe te passen in het modelleerproces, aan de andere kant is meer automatisering ook een noodzaak geworden omdat de grote hoeveelheid data niet meer handmatig verwerkt kan worden op een effectieve en efficiënte wijze.

Het onderzoek beschreven in deze dissertatie bestudeert Automatische Model Generatie (AMG). Het doel van het onderzoek was *om een methode te ontwikkelen die op automatische wijze simulatiemodellen genereert met een flexibele structuur, gebruikmakend van beschikbare data, onder de aanname dat deze simulatiemodellen bedoeld zijn voor gebruik binnen een zeker domein.* De AMG-methode die in dit proefschrift is ontwikkeld verschilt op tenminste twee manieren van eerder onderzoek naar AMG. Ten eerste zijn in de data die gebruikt wordt voor de modelgeneratie geen specificaties opgenomen van de gewenste modelstructuur. Ten tweede hebben de modellen die gegenereerd worden een structuur die dynamisch wordt bepaald tijdens het AMG-proces. Het onderzoeksdoel is uitgewerkt tot vier gerelateerde maar beter te onderzoeken onderdelen, die geadresseerd worden in de volgende vier onderzoeksvragen:

1. *Wat is een goede manier om flexibele structuren voor simulatiemodellen te definiëren die helpt om het onderzoeksdoel te bereiken?*
2. *Wat zijn de voorwaarden die aan data gesteld moeten worden om het onderzoeksdoel te kunnen bereiken?*
3. *Welke functionaliteiten moeten door een methode geboden worden om automatisch simulatiemodellen met een flexibele structuur te kunnen genereren op basis van bestaande data?*
4. *Wat is de kwaliteit van de met de methode gegenereerde simulatiemodellen?*

Voor het bestuderen van de AMG-methode is light-rail transport als toepassingsdomein gebruikt. Aangezien de methode zich richt op het genereren van simulatiemodellen voor een bepaald domein, is het noodzakelijk om een goede manier te vinden om flexibele modelstructuren voor een klasse van modellen te definiëren. In dit onderzoek hebben we hiervoor domeinspecifieke metamodellen en domeinspecifieke modelcomponenten ontwikkeld om kennis over het systeem op verschillende abstractieniveaus te representeren. Er is gebruik gemaakt van het concept van component-gebaseerd modelleren en van een geschikt onderliggend modelformalisme. Domeinspecifieke metamodellen zorgen voor de abstracte representatie van de structuur van een klasse van modellen, en de domeinspecifieke modelcomponenten specificeren het concrete gedrag van de elementen die als bouwstenen gebruikt worden voor de modelcompositie.

In lijn met bovenstaande concepten is een domeinspecifieke componentenbibliotheek genaamd LIBROS (Library for Rail Operations Simulation) ontwikkeld, gebruikmakend van het DEVS (Discrete Event System Specification) formalisme. De atomaire modellen binnen LIBROS beschrijven het gedrag van systeemelementen op het meest gedetailleerde niveau. De samengestelde modelcomponenten representeren metamodellen waarmee complexere modellen binnen het domein samengesteld kunnen worden. Ondanks de veelgehoorde bewering dat "componenten compositie ondersteunen", geven veel onderzoekers aan dat modelcompositie in de praktijk moeilijk toepasbaar is. Om modulariteit en koppelbaarheid te ondersteunen, hebben we goed gedefinieerde communicatiepoorten en samenstellingsrelaties ontworpen, evenals een gestandaardiseerd mechanisme voor communicatie binnen het model. Dit communicatiemechanisme, dat is gebaseerd op stapsgewijze doorgifte van het bericht tussen modelcomponenten, is ontworpen voor gedecentraliseerde communicatie in een netwerkmodel. Een aantal andere modelleeroplossingen worden ook besproken bij het ontwikkelen van de bibliotheek. Voorbeelden zijn het gebruik van het SAM design pattern (Shared Alterable Mixin) voor samengesteld gedrag van atomaire modellen, hergebruik en uitbreiding van modelcomponenten; het omgaan met interrupts in GDEVS (Generalized DEVS); het gebruik van Dynamic Structure DEVS (DSDEVS) voor het koppelen van de voertuigen aan de infrastructuur in het model; en gebruik van het State Update and Transition Mechanism (SUTM) met bijbehorende voordelen en complexiteit van het algoritme.

Voor de AMG-methode moeten de aanwezige gegevens voldoende concrete informatie opleveren om het model te kunnen structureren en parameteriseren. Over het algemeen zijn er echter kwaliteitsproblemen met de bestaande informatie, en zijn niet alle typen informatie aanwezig die de AMG-methode nodig heeft om de modelstructuur te kunnen opbouwen. Bij het ontwikkelen van de AMG-methode moet deze discrepantie worden geïdentificeerd, en moeten maatregelen worden genomen die het gat dichten. Binnen de AMG-methode worden voornamelijk die functionaliteiten opgenomen die automatiseerbaar zijn. Bij het genereren van simulatiemodellen voor light rail systemen wordt infrastructuurdata uit CAD-systemen gebruikt als basis voor de modelstructuur. Diverse maatregelen bleken nodig te zijn om problemen op te lossen met interne inconsistenties in de dataverzameling en volledigheid van de data. Om het probleem van volledigheid op te lossen moet een klein aantal elementen handmatig toe-

gevoegd worden aan de data, zoals de plaats waar voertuigen worden gegenereerd. Een grote uitdaging voor de AMG-methode blijft de geschiktheid van de datarepresentatie voor het ontwikkelen van de modelstructuur (waaronder het identificeren van bruikbare modelcomponenten en compositierelaties voor het model). De "geschiktheid van de datarepresentatie" is de mate waarin de gegevens geschikt zijn voor gebruik in termen van formaat, eenheid, precisie, en toereikendheid van het type. Dat laatste wordt gedefinieerd als de mate waarin de data alle typen bruikbare informatie bevat. Om het geschiktheidsprobleem op te lossen, worden de typen informatie die beschikbaar zijn in de data en die uit de data afgeleid kunnen worden, geordend op basis van hun afhankelijkheden. Op basis van deze afhankelijkheden zijn drie transformatiestappen (met verschillende sub-stappen) ontwikkeld om de benodigde data stap voor stap af te leiden en uiteindelijk simulatiemodellen te genereren.

De representaties van de datastructuren en transformaties zijn gebaseerd op de theorieën voor transformaties van grafen. Een gerichte graaf en een hypergraaf worden gecombineerd om de samengestelde modelgraaf (MGC) te vormen. Deze graaf heeft een hiërrarchische structuur, waarin (samengestelde) elementen geheel binnen één hogerliggend element opgenomen worden. Deze structuur wordt aangeduid als de Compositional Containment Hierarchy (CCH). De regels voor de transformaties van de grafen worden gespecificeerd op het niveau van het metamodel van de originele datastructuur, van de datastructuren gebruikt in de tussenstappen van de transformaties, en van het uiteindelijke simulatiemodel. De metamodellen van de data en de samengestelde datastructuren die voor de transformaties worden gebruikt, worden gedefinieerd als grafen, waarvan de overeenkomende instanties als hypertakken opgenomen worden in de MCG.

In de eerste transformatiestap wordt een gerichte (platte) graaf geconstrueerd die een afbeelding bevat van de data over de railinfrastructuur (die geen knopen bevat), gebaseerd op geometrische deducties. In de tweede transformatiestap worden delen van de grafen gematcht met patronen (met partieel geordende substappen waar zowel van compositie als van decompositie gebruik gemaakt wordt), waaruit een MCG wordt afgeleid. Voor de transformatie zijn een aantal grafenpatronen en samengestelde grafen gedefinieerd, evenals een algoritme voor het bepalen van isomorfismen in samengestelde grafen en een zoekalgoritme voor het vinden van mogelijke samenstellingen. Het doel van de MCG is het aggregeren van de knopen in de gerichte graaf naar een CCH op een zodanige manier dat de gevonden patronen in de grafen in een volgende stap getransformeerd kunnen worden naar corresponderende modelcomponenten. In de derde transformatiestap worden de knopen en de hypertakken van de MCG getransformeerd naar overeenkomstige atomaire en samengestelde modelcomponenten (inclusief hun configuratie), en de knopen worden getransformeerd naar verbindingsrelaties tussen modelcomponenten. In dit verband is de relatie tussen een hypertak (een samenstelling) en het gegenereerde gekoppelde infrastructuurmodel een injectief (eeneenduidig) homomorfisme, waarbij een aantal subcomponenten zijn toegevoegd aan het resulterende infrastructuurmodel conform de definities in het metamodel. Een voorbeeld van een toegevoegde component is de besturingslogica van een kruispunt, die dynamisch wordt geconfigureerd. Na afloop van deze transformatiestappen be-

schikken we over een simulatiemodel waar het gedrag op elementair niveau is voorgedefinieerd in de atomaire modelcomponenten van de domeingerelateerde bouwsteenbibliotheek, en de modelstructuur dynamisch is opgebouwd uit samengestelde componenten op grond van de beschikbare infrastructuurgegevens.

Na afloop van de modelgeneratie kan het model worden gekalibreerd met behulp van kwantitatieve data afkomstig uit het afgebeelde systeem, voor zover dit voorhanden is. De kalibratieprocedure gebruikt speciaal gedefinieerde goodness-of-fit testen en zoekmethoden voor parameters. Voor de kalibratie is een eerste onderzoek uitgevoerd om rijtijden tussen haltes te schatten voor een gegenereerd LIBROS-model van de Gemeente Den Haag. Verschillende observaties en problemen die te maken hebben met de kalibratie worden besproken in het proefschrift. De kwaliteitstoets die uitgevoerd wordt voor de kalibratie kan natuurlijk ook (met een complementaire dataset) gebruikt worden voor het valideren van de werking van het model.

Gedurende het onderzoek en het ontwikkelen van de methoden en de resulterende modellen is gebruik gemaakt van richtlijnen en bewezen werkwijzen uit het vakgebied Modelbouw en Simulatie om valide en bruikbare modellen te bouwen. Als een afsluitende validatiestap in het onderzoek en als een methode om de AMG-methode te evalueren, is een validatiesessie (met enquêtes en interviews als dataverzamelingsmethodes) uitgevoerd met negen inhoudsdeskundigen. Er is specifiek gevraagd om het modelgedrag en de interacties tussen modelcomponenten te evalueren om de structurele validiteit van de gegenereerde LIBROS light-rail-modellen van de Gemeente Den Haag vast te stellen. Op basis van de validatiesessies zijn kleine aanpassingen gemaakt in de modellen en in de parameters voor het genereren van modellen, zodat de AMG-resultaten (nog) beter aansluiten bij de inzichten van de experts. Geen van de experts heeft in de validatiesessie aangegeven dat er structurele problemen waren met de gegenereerde modellen. Over het algemeen waren de experts zeer tevreden over het feit dat complexe simulatiemodellen automatisch gegenereerd kunnen worden op basis van infrastructuurgegevens, en ze ondersteunden de validiteit van de modellen voor simulatiestudies zoals dienstregelingsonderzoek en analyse van rijtijden. De experts hadden verschillende suggesties voor vervolgonderzoek.

Het onderzoek heeft, gebruikmakend van cases in light-rail transport, een methode opgeleverd die op geautomatiseerde wijze valide simulatiemodellen met flexibele structuren kan genereren op basis van bestaande infrastructuurgegevens en een bouwsteenbibliotheek. De bibliotheek is ontworpen om modulariteit en samenstelbaarheid van de componenten te ondersteunen. Het bevat domeinspecifieke modelcomponenten voor de AMG-methode waarmee concreet modelgedrag op elementair niveau gespecificeerd kan worden. Daarnaast zijn componenten gedefinieerd die metamodellen voor het domein beschrijven, waarmee complexere structuren dynamisch samengesteld kunnen worden bij het toepassen van de AMG-methode. Om het probleem op te lossen dat beschikbare gegevens veelal onvolledig en maar beperkt geschikt zijn voor het genereren van modellen, zijn transformatieregels voor de data ontwikkeld. De transformaties maken gebruik van grafen en samenstellingen van grafen. Een incrementeel algoritme matcht achtereenvolgens de patronen in de grafen met patronen in de data,

waarbij een deel van de graaf bij een match vervangen wordt door een samengestelde graaf, op grond waarvan uiteindelijk de structuur van het simulatiemodel wordt gegenereerd.

Over het algemeen is een goed ontwerp van een flexibele modelstructuur de uitkomst van een groot aantal overwegingen: een generiek modelformalisme, formele definities van modelstructuren, scheiden van verschillende belangen, en herbruikbare definities van modelgedrag, modelkoppelingen, modelinteracties, etc. De transformatieregels voor AMG kunnen worden gedefinieerd onder de conditie dat de beschikbare gegevens voor AMG semantisch en pragmatisch compleet zijn, dat de syntactische- en afbeeldingsinconsistenties meetbaar zijn, en dat de modelleurs over voldoende domeinkennis beschikken om de transformatieregels te definiëren die problemen met volledigheid en geschiktheid van de beschikbare gegevens moeten oplossen. De transformatieregels worden gedefinieerd op basis van de metamodellen van de originele datastructuren, de tussenliggende datastructuren en het gegenereerde simulatiemodel. Hierdoor kunnen de elementen in de representatie van de modelstructuur afgebeeld worden op de overeenkomstige voorgedefinieerde simulatiemodelcomponenten, en kan een werkend simulatiemodel gespecificeerd worden waarvan de structuur overeenkomt met de modelstructuur die afgeleid is van de oorspronkelijke data.

De AMG-methode bouwt voort op systeemtheorie, theorie over modelbouw en simulatie, en grafentheorie waaronder transformaties van grafen en hypergrafen. De methode (en de gegenereerde modellen) zijn getest in het domein van light-rail transport, maar kunnen in een bredere veld toegepast worden. Wanneer de structuur van een systeem voorgesteld kan worden als een graaf, zoals bij bijna alle systemen in het infrastructuurdomein, lijken er goede kansen voor gebruik van de methode te zijn. Verder onderzoek zal moeten uitwijzen of dat inderdaad het geval is. Daarnaast kan nader onderzoek worden gedaan naar calibratietechnieken en naar generatie van multiresolutie- en multi-perspectiefmodellen.

One of today's challenges in the field of modeling and simulation is to model increasingly larger and more complex systems. Complex models take long to develop and incur high costs. With the advances in data collection technologies and more popular use of computer-aided systems, more data has become available in many organizations. This often allows for and requires a certain degree of automation in modeling. The research presented in this dissertation studied how to automatically generate simulation models. The method proposed uses domain specific model components as building blocks for model generation and applies graph transformation based algorithms to compose large simulation models according to the existing data. The method has been applied practically in the domain of light-rail transportation.

ISBN 978-94-6203-461-7
90000 >
9 789462 034617