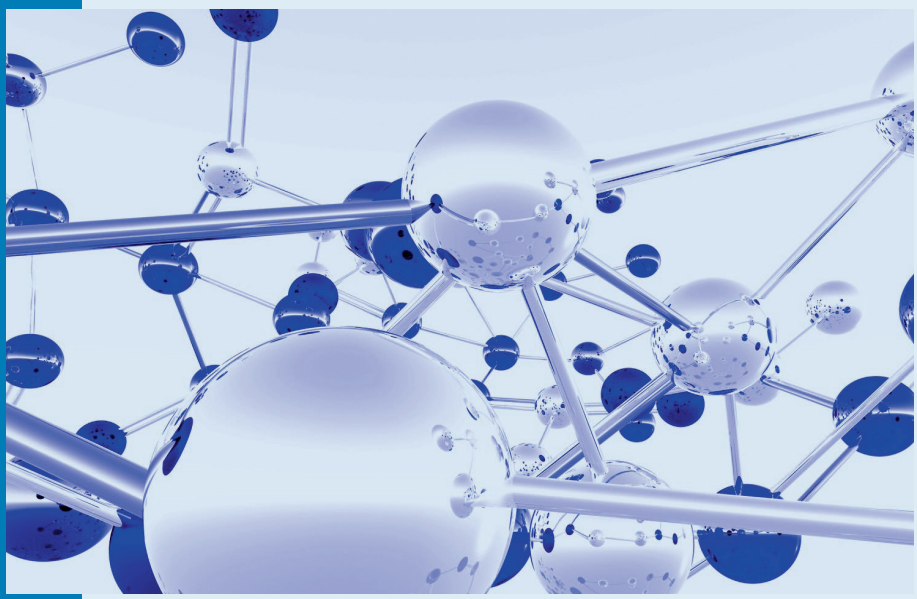
Model Driven Development of Simulation Models

Defining and Transforming Conceptual Models into Simulation Models by Using Metamodels and Model Transformations



Deniz Çetinkaya

Model Driven Development of Simulation Models

Defining and Transforming Conceptual Models into Simulation Models by Using Metamodels and Model Transformations

Model Driven Development of Simulation Models

Defining and Transforming Conceptual Models into Simulation Models by Using Metamodels and Model Transformations

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 maandag 4 november 2013 om 15.00 uur
door

Deniz ÇETİNKAYA

Master of Science in Computer Engineering
Middle East Technical University
geboren te Konya, Turkije.

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. A. Verbraeck

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. A. Verbraeck	Technische Universiteit Delft, promotor
Dr. M.D. Seck	Technische Universiteit Delft
Prof. dr. F.M.T. Brazier	Technische Universiteit Delft
Prof. dr. ir. P.M. Herder	Technische Universiteit Delft
Prof. dr. E. Visser	Technische Universiteit Delft
Prof. dr. ir. H. Vangheluwe	University of Antwerp
Prof. dr. A. Tolk	Old Dominion University
Prof. dr. ir. M.F.W.H.A. Janssen	Technische Universiteit Delft, reservelid

ISBN/EAN 978-94-6108-513-9

To my parents, my husband, and my children.

Published and distributed by:

Deniz Çetinkaya

Systems Engineering Section
Faculty of Technology, Policy and Management (TBM)
Delft University of Technology
Jaffalaan 5, 2628BX Delft, The Netherlands
Phone: +31-15-2788380
email: d.cetinkaya@tudelft.nl, cetinkaya_deniz@yahoo.com

Printed by: Gildeprint Drukkerijen - The Netherlands

Doctoral Dissertation, Delft University of Technology, The Netherlands

Contents

List of Figures

List of Tables

Acknowledgements

This research study and doctoral dissertation could not have been completed without the help and support of many people, who I hope to acknowledge in this section.

First and foremost, I would like to thank my promotor Prof.dr.ir. Alexander Verbraeck and my daily supervisor Dr. Mamadou D. Seck for their great guidance, support and motivation throughout my research. I am extremely grateful to Alexander, his outstanding vision and strong work ethic have been a great source of inspiration for me. Similarly, I deeply appreciate Mamadou's insightful suggestions and warm encouragements throughout the completion of this thesis.

Secondly, I would like to extend my gratitude to all my present and past colleagues in the Systems Engineering section. I would like to especially thank Prof.dr. Frances M.T. Brazier for her future directions and encouragements. I also would like to thank Dr. Joseph Barjis for his valuable suggestions and comments. Special thanks to Çağrı Tekinay, Yakup Koç, Shalini Kurapati, Mingxin Zhang, Rens Kortmann, Martijn E. Warnier, Tanja Buttler, Farideh Heidari, Yilin Huang, Kassidy Clark, Jos L.M. Vrancken, Stephan Lukosch, Gwendolyn Kolfschoten, Sander van Splunter, Heide Lukosch and Job Honig for providing me a friendly environment as well as for many useful discussions. I would like to especially thank Jos L.M. Vrancken for his help with the Dutch translation of the thesis summary. I also would like to thank Igor Rust for his willingness to test my prototype and help with the implementation. Many thanks to Diones G. Supriana, Everdine M.C. de Vreede-Volkers and Sabrina Ramos Rodrigues for their great administrative support.

I would like to thank the people who participated in my research, who contributed either through the emails or meetings and whose insights and views shaped my work. First of all, I would like thank Dr. Saurabh Mittal and Dr. José L. Risco-Martín for giving me the opportunity to write two chapters for their book. It was a great experience for me. I also want to thank Dr. Hessam Sarjoughian, Prof. Gabriel Wainer, Dr. Alfredo Garro, Dr. Andrea D'Ambrogio and Dr. Ola Batarseh for the discussions that we had during Spring Simulation Multi-Conference. Besides, special thanks to Prof. Osman Balcı for meeting with me during his visit to our department in 2009. It was my first days at TUDelft and he gave me the motivation and encouragement that I needed. I also want to thank Dr. Levent Yılmaz for his valuable suggestions and comments about my research.

I want to thank the committee members outside the Systems Engineering section, consisting of Prof. Hans Vangheluwe (University of Antwerp), Prof. Andreas Tolk (Old Dominion University), Prof. Paulien Herder (Infrastructure Systems and

Services Department, TUDelft), Prof. Eelco Visser (Department of Software and Computer Technology, TUDelft) and Prof. Marijn Janssen (Infrastructure Systems and Services Department, TUDelft) for their time to review my thesis. It is an honor for me to see you in my committee.

Finally, I take this opportunity to express my profound appreciation to my family. I want to thank my mother Gülşen Küçükkeçeci and my father Muhsin Küçükkeçeci for their unlimited love and endless patience. This success would not have been possible without you. I love you so much. *(Canım Annem ve Babam, sınırsız sevginiz ve bitmeyen sabrınız için çok teşekkür ediyorum. Sizin desteğiniz olmasaydı, bu günlere gelemez ve bu başarıları elde edemezdim. Sizi çok seviyorum.)*

I owe many thanks to my brothers, Onur and Cihan, for their understanding and support while I have been abroad for more than 8 years. You have been always with me although thousands kilometers away. *(Sevgili kardeşlerim Onur ve Cihan, yurt dışında geçirdiğim 8 yıldan fazla süredir göstermiş olduğunuz anlayış ve destek için size çok teşekkür borçluyum. Aramızdaki mesafelere rağmen hep benimleydiniz. Her zaman yanınızda olamadım ama aklımdan hiç çıkmadınız.)*

My deepest thanks go to my children, Yıldız and Hakan, for being so patient and helpful when I have been so busy during my studies. I love you very much. *(Biricik çocuklarım Yıldız ve Hakan, doktora çalışmalarım sırasında çok yoğun olduğum zamanlarda göstermiş olduğunuz anlayış ve sabır için size çok teşekkür ediyorum. Sizi çok seviyorum.)*

And, my wonderful husband Orhan. Your love, trust and support enabled me to complete this thesis. Thank you for being with me all the way. My life is beautiful and meaningful only with you.

Deniz (Küçükkeçeci) Çetinkaya
Delft, June 2013

Chapter 1

Introduction

Modeling and simulation is an effective method for analyzing and designing systems and it is of interest to scientists and engineers from all disciplines [Pid02]. Simulation is the process of conducting experiments with a model for a specific purpose such as analysis, problem solving, decision support, training, entertainment, testing, research or education [Sha75, Bal01]. The fundamental prerequisite for simulation is a model, which is called a simulation model. A simulation model is developed through a modeling process, which is called simulation model development. Therefore, the activities in a simulation study are collectively referred to as Modeling and Simulation (M&S) [Bal01]. Simulation models are typically built for individual projects and very little advantage is taken from existing models developed earlier [KN00]. Thus, redundant representations of the same concepts are often developed for simulation projects.

Several methodologies have been proposed to guide modelers through various stages of M&S and to increase the probability of success in simulation studies [Sha75, RAD+83, Ban98, Bal12]. Each methodology suggests a body of methods, techniques, procedures, guidelines, patterns and/or tools as well as a number of required steps to develop and execute a simulation model. Most of the well known modeling and simulation methodologies state the importance of conceptual modeling in simulation studies and they suggest the use of conceptual models during the simulation model development process. However, the transformation from a conceptual model to an executable simulation model is often not addressed [vdZKT+10]. Besides, none of the existing modeling and simulation methodologies provides guidance for formal model transformations between the models at different abstraction levels. As a result, conceptual models are often not used explicitly in the further steps of the simulation studies and a big semantic gap exists between the different models of the simulation projects [Rob06, BAO11]. Because of this gap there is not an easy way of tracing the concepts in different models.

From the software engineering perspective, a (computer) simulation model can be seen as a software application and an M&S study can be seen as a software engineering project, as a simulation model is an executable program written in a programming language. The programming language can be either a general pur-

pose programming language (such as C++, Java, etc.) or a specialized simulation programming language (such as SIMSCRIPT, SIMAN, SIMULA, etc.). In both cases, an interpreter executes the simulation model. Thus, software engineering methodologies can be applied to M&S and existing tools and techniques can be utilized.

In order to address the identified issues in the M&S field, this research suggests the application of a Model Driven Development (MDD) approach throughout the whole set of M&S activities and it proposes a formal MDD framework for modeling and simulation. MDD is a software engineering methodology that suggests the systematic use of models as the primary means of a development process [KWB03]. MDD introduces model transformations between the models at different abstraction levels and suggests the use of metamodels for specifying modeling languages. In MDD, models are transformed into other models in order to (semi)automatically generate the final (software) system. MDD has been proposed to improve productivity, maintainability, and quality during a development process [AK03, Sel06, KJB+09, PTTT09, MCM13]. Due to the similarities between software development and simulation model development, MDD could potentially be a cost and effort saving approach for the M&S research and practices. Applying an MDD approach in simulation could in principle reduce the gap between the conceptual modeling and the simulation model development stages, and increase the quality of the simulation study.

The outline of this chapter is as follows: The following three sections provide some background information about systems engineering, M&S, and M&S lifecycles. The selected research issues in the M&S field are explained in Section 1.4. The research objective and the research questions are presented in Section 1.5. The research philosophy and the research strategy are explained in Section 1.6 and 1.7 respectively. The outline of the thesis is given in Section 1.8.

1.1. Systems thinking

Thinking in terms of systems is a way of looking at the world and the term system has been used for centuries with various meanings in many fields. A typically used definition of a system is "a set of interrelated components working together toward some common objective or purpose" [Kli69, SA00, KS03, BF06]. Producing system-level purposeful results is the basic characteristic of a system. A system may be classified as a natural or human-made system; physical or conceptual system; closed-loop or open-loop system; static or dynamic system, and so on [BF06]. Management systems, transportation systems, health care information systems, economic systems, education systems, manufacturing systems, military systems, biological organisms, electronic systems, hardware systems, organizations, social systems, etc. are all examples of systems with various complexities [KS03, Wym93].

Systems thinking provides a rigorous way of understanding and expressing real world situations based on a part-whole hierarchy of components. In this way, it helps to

deal with complexity [vB68, vG91]. The components can include people, hardware, software, facilities, policies, documents, or other things which are required to achieve the system purpose. The components of a system can be other subsystems as well [Bun79]. Each component of the system has a number of properties and functions. Besides, a system has its own properties [Che99]. The system-level results are produced by its components over time through the relationships between them [Ash57, SA00]. Hence, the value added by the system as a whole is beyond that contributed independently by its parts. A system is bounded within an environment. The environment of a system is a set of components and their relevant properties, which are not part of the system but a change in any of which can cause or produce a change in the state of the system [AE72].

Systems engineering is an interdisciplinary field based on the systems thinking that manages, guides or supports engineering projects to enable the realization of successful systems [INC06]. The main purpose of systems engineering is to achieve a high standard of overall quality, performance and reliability of an engineering project [SA00].

Systems engineering is a comprehensive activity, and so system engineers utilize different methods and techniques such as modeling and simulation, functional analysis, hardware in-the-loop testing, reliability analysis, etc. [NAS07]. Systems engineering methods can be applied at any stage of the project lifecycle such as analysis, design, development, test, operation, integration, update, or management. This research is about modeling and simulation. M&S is an important and useful tool, which enables study of the dynamic behavior of a system. M&S is used for many years in various fields such as business, economics, marketing, education, politics, social science, transportation, international relations, urban studies, global systems, etc. [Pid02, Sha75]. The next section presents more information about M&S.

1.2. Modeling and simulation

Simulation is the process of conducting experiments with a model for a specific purpose [Sha75, Bal01]. Broadly speaking, a model is a representation of something. The represented thing is called the source which can be for instance an object, an idea, a phenomenon, an organization, a process or an event. A model can even be a representation of another model. From the systems perspective, a model is a representation of a system and the represented thing is called the source system [ZPK00].

A simulation model is a representation of a system which can be simulated by means of experimentation [Kle08]. It may be a physical model, a formal (mathematical) model, a computer model, or a combination of these [RAD+83]. A wind tunnel, a wave tank or a scaled down model of a plane can be examples of physical simulation models. Since physical models are often relatively expensive to build, formal models are preferred in many cases. If the calculations have to be performed by hand in

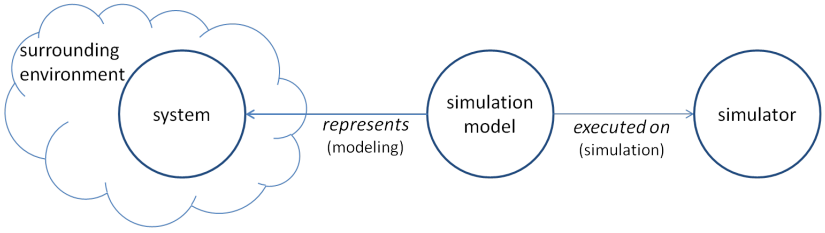


Figure 1.1: Basic concepts in modeling and simulation.

a formal model, simulation can be extremely tedious and costly. Due to the rapid growth of the computer technology, computer simulation has replaced simulation using hand calculations, and computer simulation models are used in many fields. A simulation model is executed on a simulation platform to generate simulation results, which is generally called as simulator [ZPK00]. If not stated otherwise, a simulation model refers to an executable computer model in the context of this thesis. Figure 1.1 shows the basic concepts and relations in M&S.

Models are commonly classified according to how they deal with time, randomness and state as dynamic/static models, deterministic/stochastic models and discrete/ continuous models respectively [LK91, Nan81].

- *Dynamic vs. static models:* A dynamic model represents a system as it evolves over time; whereas a static model is a representation of a system at a particular time, or one that may be used to represent a system in which time simply plays no role.
- *Deterministic vs. stochastic models:* In deterministic models, a model does not contain any probabilistic elements; and the output is determined once the set of input values and relationships in the model have been specified, even though it might take a lot of computer time to evaluate what it is. Stochastic models produce output using probabilistic or random variables.
- *Discrete vs. continuous models:* In continuous models, state variables change continuously with respect to time. In discrete models, state variables change instantaneously at distinct points in time.

Simulation models can also be classified along these three dimensions [LK91]. However, in simulation studies, the source system has some sort of mechanism that changes [Wym67], and so the simulation models are dynamic in nature. On the other hand, deterministic simulation models are seen as a special case of stochastic models, since their parameters or input variables are generally sampled from a given prior distribution [Kle08].

1.3. M&S lifecycles

Several methodologies have been proposed in the literature to guide modelers through various stages of M&S and to increase the probability of success in simulation studies [Sha75, RAD+83, Ban98, Bal12]. Looking at the existing methodologies, some similar terms and patterns can be distinguished. Although the concept of a lifecycle has not been fully employed in the field of M&S [Bal12], an M&S lifecycle with five main stages can be characterized; (1) M&S study definition, (2) conceptual modeling, (3) simulation model development, (4) experimentation and (5) analysis of results. In this section, a set of M&S lifecycles will be analyzed in order to identify the common steps and concepts among them. The activities are labeled with the numbers 1 to 5 to show which activities fall into the proposed main stages. Original names of the stages are given in the parentheses if they are comparable to the given ones.

In Shannon's methodology [Sha75, Sha98], a simulation study starts with defining the problem and determining the boundaries (1). After deciding to use simulation, abstraction of the real system is represented with a flow diagram and the data needed by the model is identified in an appropriate form (2). Then the model is described in a computer programming language and validated (3). After that strategic and tactical planning for the experiments are done; and simulation model is executed to generate the results to perform analysis (4). At the end, results are analyzed and the model and/or results are implemented while the activities and outcomes are documented (5). Implementation here means putting the model and/or results to use.

Roberts et al. [RAD+83] define a similar lifecycle, which includes six stages to construct a computer simulation model. The first stage involves recognizing and defining a problem to study (problem definition) (1). The second stage involves committing to paper the important influences believed to be operating within a system (system conceptualization) (2). After that, models are represented in the form of computer code that can be executed (model representation) (3). In the fourth stage, computer simulation is used to determine how all of the variables within the system behave over time (model behavior) (4). In the model evaluation stage, numerous tests must be performed on the model to evaluate its quality and validity (model evaluation). Lastly, the model is used to test alternative policies that might be implemented in the system under study (policy analysis and model use) (5).

Nance [Nan84, Nan94] characterizes the model lifecycle as problem definition, model development, and decision support. The problem definition is dependent on both technical and organizational factors, and success can be achieved only by effective communication among the participants and the documentation of decisions reached during these stages (1). Model development starts by expressing the mental model in the minds of one or more modelers (he calls it a conceptual model) in the form of one or more communicative models (a communicative model

can be compared to a conceptual model in the context of this thesis) (2). Then, the program model follows from a communicative model (3); and, embodied within an experimental design, the experimental model produces results (4). The integrated decision support period is initiated with the acceptability of the model (5). Again, both technical and organizational factors can contribute to the acceptance decision. Besides, Nance states the importance of tool support throughout the model lifecycle.

Fishwick [Fis95] identifies simulation as a tightly coupled and iterative process composed of model design, model execution, and execution analysis. He focuses on model design and model execution and explains the steps to develop and use a simulation model. He defines the first step as gathering data associated with the system (1). Then, from the data and knowledge of past experiments with similar systems, a model is formulated. The model will usually contain parameters which need to be initialized to some specific value (2). After that, models must be converted to algorithms to run on a digital computer (3). Verification is the process of making sure that the written computer program corresponds precisely to the formal model. Validation is the process of making sure that the model's output accurately reflects the behavioral relationships present within the source system data. The model, when simulated, should be able to produce the same sorts of data and input output relationships that were gathered initially (4). Lastly, analysis tests on the data generated from the model are performed (5).

Banks [Ban98] defines a lifecycle with twelve steps which is similar to Shannon's. The lifecycle can be broken down into four main stages. The first stage consists of problem formulation and setting of objectives and overall design. The initial statement of the problem is usually not well defined and the initial objectives usually need to be refined (1). The second stage is related to model building and data collection and includes model conceptualization, data collection, model translation, verification, and validation (2-3). The third stage concerns running the model. It involves experimental design, simulation model runs and analysis (4-5). This stage must have a thoroughly conceived plan for experimenting with the simulation model. The fourth stage involves reporting and implementation (5).

Law [Law03] defines a seven step approach for conducting a successful simulation study. First, the problem of interest is stated by the decision maker (1). Then the conceptual model is prepared and a structured walk through of the model is performed for conceptual model validation (2). After that, the conceptual model is programmed in either a general purpose programming language or in a commercial simulation software product. The programmed model is validated according to the comparable performance measures collected from the actual existing system (3). Once the simulation model is ready, simulation experiments are designed and executed (4). The results are analyzed and additional experiments are performed if needed. Lastly, the documentation for the model and the associated simulation study are presented (5).

Robinson [Rob04] gives special attention to conceptual modeling and presents a lifecycle with four stages. The first stage includes the understanding of the problem situation and data collection (conceptual modeling) (1-2). In the second stage, the conceptual model is converted into a computer model (model coding). This simply refers to the development of the model on a computer (3). Once developed, experiments are performed with the simulation model in order to obtain a better understanding of the real world and/or to find solutions to real world problems (experimentation) (4). The fourth stage is implementation. Implementation can be thought of in three ways. First, it is implementing the findings from a simulation study in the real world. Where the simulation study has identified a particular solution to a real world problem, then implementation is a case of putting this solution into practice. A second interpretation of implementation is implementing the model in the real world rather than the findings. The third interpretation is implementation as learning. Where the study has led to an improved understanding, implementation is less explicit, but should be apparent in future decision making. These forms of implementation are not mutually exclusive and a simulation study might result in two or even three of these types (5).

A recent work of Balci [Bal12] introduces a more detailed and comprehensive M&S lifecycle. The lifecycle consists of processes, work products, verification and validation activities, quality assurance activities, and project management activities required to develop, use, maintain, and reuse an M&S application from birth to retirement. The main processes in the lifecycle are problem formulation, requirements engineering (1), conceptual modeling (2), architecting, design, implementation (3), integration, experimentation (4), presentation (5) and certification.

There have been attempts to standardize the M&S activities. For example, in the military domain, HLA Federation Development and Execution Process (FEDEP) (or the newer version DSEEP [IEE10]) is proposed, which provides a process for developing interoperable HLA based federations [IEE03]. A federation development can be compared to a simulation model development [MNA06, KCG+08]. FEDEP presents seven main stages as: defining federation objectives (1), performing conceptual analysis, designing the federation (2), developing the federation (3), planning, integrating and testing the federation, executing the federation and preparing the results (4), and analyzing output data and evaluating results (5).

Many other lifecycles are proposed to outline the key processes in simulation studies [Rob04]. However, when the lifecycles are analyzed in detail, similar steps can be recognized. The main differences lie in the naming of the processes and the number of the stages into which they are split [Rob04]. Please note that the lifecycles are compared just in order to obtain a common terminology. A possible generic modeling and simulation lifecycle with five main stages is presented in Figure 1.2. As a summary,

1. In the M&S study definition stage, the problem or issues are identified and the purpose of the simulation study is stated. The requirements are defined

and the outline of the M&S study is presented in the M&S plan.

2. In the conceptual modeling stage, a conceptual model is developed while the system is investigated and data is gathered.

3. In the simulation model development stage, an executable simulation model is developed according to the conceptual model. Formulating a mathematical model first, i.e. a formal model, is suggested at this stage. Model validation and verification is done according to the collected data before executing the model.

4. In the experimentation stage, experiments are designed and the simulation model is executed on a simulator to generate simulation results.

5. In the analysis stage, the experimentation results are analyzed and results are presented in a report. Additional experiments can be performed if needed.

An iterative approach can be followed between the development stages. Model validation deals with building the right model and it is used to determine that a model is an accurate representation of the source system. During the model validation, the model behavior compared with respect to the source system behavior. Model verification deals with building the model right and it is used to ensure that the model is developed correctly according to the modeling method and it functions properly without error. During model verification, the accuracy of the model transformation from one form into another is tested as well. The final implementation or model use stage [Sha75, RAD+83, Rob04] is excluded from this lifecycle since it is assumed to be in the scope of the systems engineering lifecycle [SA00].

1.4. Identified issues in the M&S field

When the outputs of the M&S lifecycle are examined, it can be easily noticed that the development process relies on models at different stages. At least four models are developed throughout the M&S lifecycle, which are conceptual model, formal model, computer model and experimental model. Although the definition of a computer model is clear and the relationship between a computer model and an experimental model is defined through the design of experiments, there is not a clear understanding of how a conceptual model and a formal model relate to a computer model and to each other.

Recent studies state the importance of conceptual modeling in simulation studies [YO06, Rob06, BD08, RBKvdZ10]. The most important role of a conceptual model is to make all parties involved in a simulation project to understand the models in the same way [KR08]. Proper development of a simulation conceptual model is critical for expressing the objectives of the simulation study. Surprisingly there are many simulation projects that have no conceptual model, a poorly or only partially developed conceptual model, or incomplete documentation of the

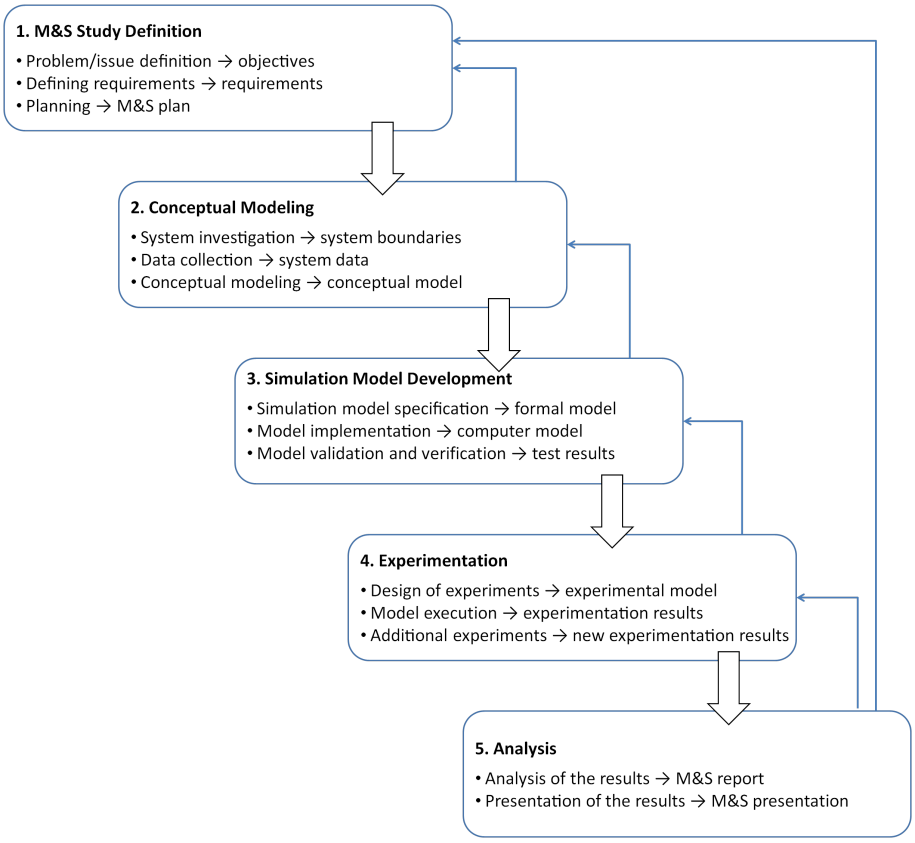


Figure 1.2: A generic M&S lifecycle.

simulation conceptual model [Pac00, RBKvdZ10]. For an effective conceptual modeling stage, more formal and precise methods and tools are needed that will enable the explicit use of conceptual models [Pac00].

Issue 1. *There is a lack of tool support that will enable the explicit use of the conceptual models in simulation studies.*

Despite the fact that most of the existing modeling and simulation methodologies suggest the use of conceptual models before the simulation model development process, a very small amount of them refer to how to move from a conceptual model to an executable simulation model [vdZKT+10]. But none of them provides a formal method for model transformations between the models at different stages. As a result, there is no guidance for formal model transformations while moving from a conceptual model to an executable simulation model.

Issue 2. *There is no commonly accepted guidance for formal model transformations between the different models of the M&S lifecycle.*

Due to conceptual models are often not used explicitly in the further steps of the M&S lifecycle, a big semantic gap exists between the different models of the simulation project. This gap causes a lack of model continuity in many cases. Model continuity refers to the generation of an approximate morphism relation [Far98, ZPK00, RB04] between the different models of a development process through predefined transformation rules. Model continuity is obtained if the initial and intermediate models are effectively consumed in the later steps of a development process and the modeling relation is preserved [HZ05].

The lack of model continuity has a potential risk of increased design and development costs due to unnecessary iterations [JWLBB02]. In the current M&S practice, model continuity can only be obtained implicitly by the simulation modeler, which is hard to assess. If model continuity can be guaranteed explicitly then it can increase the maintainability and quality of a simulation study, and it can decrease the risk of failure [HZ05].

Issue 3. *There is a model continuity problem throughout the M&S lifecycle.*

1.5. Research objective and questions

The objective of this research is to design a framework for M&S that would provide a set of methods and guidelines for specifying (conceptual) models in a well-defined manner (to address issue 1), for performing formal model transformations on those models (to address issue 2), and for supporting model continuity throughout the M&S lifecycle (to address issue 3). To address the identified issues in the M&S field and to achieve the research objective, two main research questions and a number of subquestions are investigated throughout this research.

Question 1. *How can we support the conceptual modeling stage in M&S?*

- **Q.1.1.** What are the requirements for an effective conceptual modeling stage in M&S?
- **Q.1.2.** How can a conceptual modeling language help to meet these requirements?

Question 2. *How can we provide model continuity throughout the M&S lifecycle?*

- **Q.2.1.** How can simulation conceptual models be utilized in the further steps of the simulation study?
- **Q.2.2.** How can formal model transformations help to bridge the gap between the different models in the M&S lifecycle?

In the first question, the focus of the research is on supporting the conceptual modeling stage in M&S. So, the requirements for an effective conceptual modeling stage needs to be examined first (Q.1.1). After that, the formal usage of conceptual modeling languages to meet these requirements is researched (Q.1.2). In the second question, the focus of the research is on providing model continuity throughout the M&S lifecycle. Hence, a practical and formal way to utilize simulation conceptual models in the further steps of the simulation study needs to be searched first (Q.2.1). After that, the usage of formal model transformations to bridge the gap between the different models in the M&S lifecycle is researched (Q.2.2).

To answer the research questions, this research proposes the application of an MDD approach throughout the M&S lifecycle. From the software engineering perspective, a (computer) simulation model can be seen as a software application and an M&S study can be seen as a software engineering project, as a simulation model is an executable program written in a programming language. Thus, we believe that software engineering methodologies can be applied to M&S and existing tools and techniques can be utilized. MDD introduces model transformations between the models at different abstraction levels and suggests the use of metamodels for specifying modeling languages. Applying an MDD approach in simulation could in principle reduce the gap between the conceptual modeling and the simulation model development stages, and increase the effective use of conceptual models. Based on our observations and background research, we formulate the following hypothesis:

The use of the MDD methods, techniques and tools can improve the conceptual modeling stage in simulation studies and provide model continuity between the different models in the M&S lifecycle.

Background information that is used to formulate the hypothesis is given in Chapter 2. To test this hypothesis, a formal MDD framework for M&S is proposed and a case example is performed according to the framework. The following sections present the research philosophy, research strategy and outline of the thesis.

1.6. Research philosophy

Research in its most general form is a systematic inquiry to understand existing knowledge and generate new knowledge [GKV01]. A research philosophy relates to the development of knowledge and contains assumptions about the way in which a researcher views the reality and knowledge [SLT07]. It guides the researcher in choosing the research strategy and appropriate methods for conducting successful research. Research philosophy is built on three major components which are ontology, epistemology and methodology, whereas there is a close relationship between them.

Ontology refers to the nature of reality. The dominant ontological paradigms are realism, critical realism, pragmatic realism and idealism [LT08]. Realism merely assumes that there is some sort of reality which is independent of the observer [Hol97]. Critical realism assumes that there is an imperfectly understandable reality. Claims about reality are subjected to the widest possible critical examination to facilitate apprehending the reality as closely as possible [GL94]. Pragmatic realism acknowledges the idea of relativity while defending the moderately realist view [LT08]. It accepts that some part of the reality can be dependent on some other parts in a specified context. Thus, pragmatic realism accepts a conceptual system, which may be real for some people, but not for others. Idealism, in contrast to realism, assumes that the reality is purely dependent on the activity of mind and it is purely an observer's perception [Sta08]. In this research, it is believed that although there is some sort of reality which is independent of the observer, it is still not possible to perfectly interpret and understand it. Hence, critical realist ontology is chosen in this research.

Epistemology refers to the theory of knowledge and the major paradigms are positivism, postpositivism, pragmatism and interpretivism [Hol97]. Epistemological paradigms depend on the beliefs about the nature of knowledge. Positivism believes that all knowledge about reality is objectively given and observer is capable of studying it without influencing it [GL94]. Postpositivism tries to be less certain regarding to claims about reality, but it is still closely related to positivism [TT98]. It states that the apprehension of the reality can only be imperfect and incomplete. Pragmatism believes that the knowledge is relative to the overall goals and objectives of the observer. Thus, it is more teleological or goal oriented. Interpretivism believes that all knowledge about reality is only constructed and depends on human perception and experience. Constructions are not more or less true, but more or less sophisticated [OB91]. In this research, it is believed that reality can be agreed upon by independent observers while being respectful to the idea of relativity. So, this research, from a philosophical point of view, is posited on a postpositivist epistemology.

Methodology refers to the selection of an appropriate set of research methods which will be used during research [OB91]. There are a wide variety of research methods to gain and enhance knowledge; and it is possible to categorize them in many ways.

A commonly used categorization is according to the nature of data, as quantitative and qualitative. Quantitative approach is concerned with the collection and analysis of measurable and statistic data. It tends to emphasize relatively large scale and representative sets of data. Qualitative approach, on the other hand, is concerned with collecting and analyzing information not subject to numeric measurement in as many forms as possible [BHT96].

Quantitative methods include survey methods, formal methods for data analysis, laboratory experiments and numerical methods such as mathematical modeling, among others. Qualitative methods include the observation, participant observation and interviews, among others. This research uses mostly quantitative methods.

Another categorization of the methods is done according to the underlying type of reasoning such as deductive reasoning, inductive reasoning and abductive reasoning [Yu06]. Reasoning is the process of using existing knowledge to draw conclusions, make predictions, or construct explanations.

- *Deductive reasoning* is a method of reasoning in which general principles or premises are proceeded to derive particular information as deductive arguments. (*e.g. All birds have feathers. Cino is a bird. Therefore, Cino has feathers.*)
- *Inductive reasoning* is a method of reasoning in which the premises of an argument indicate some degree of support for the conclusion but do not ensure it. (*e.g. All of the sugar we have examined so far is sweet. Therefore, all sugar is sweet.*)
- *Abductive reasoning* is a method of reasoning in which a set of accepted facts is proceeded to infer to the best explanation for the relevant evidence. (*e.g. All the beads from this box are blue. These beads are blue. Therefore, these beads are from this box.*)

Deductive reasoning based research tests or evaluates a hypothesis, while inductive or abductive reasoning based research generates or suggests a hypothesis. The underlying reasoning in this research is deductive reasoning, such that the hypothesis given in Section 1.5 is tested throughout the research. An overview of the research paradigms is given in Figure 1.3 to provide a comprehensive view. In many cases method selection depends on the research questions, research objective, available time, research funds and researcher's background as well as the ontological and epistemological position. In order to achieve the objective of this research, a number of suitable research methods have been used which are listed below:

- *Literature review:* Scientific reading to gather background information.
- *Prototyping:* Testing the applicability of a theory by a proof of concept implementation.

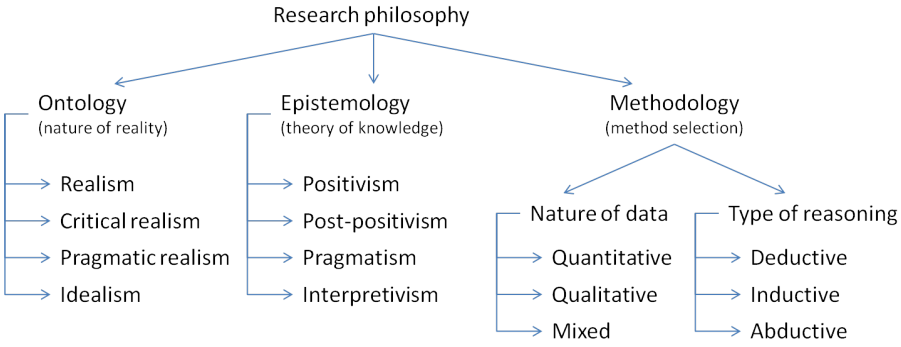


Figure 1.3: Research paradigms from the ontological, epistemological and methodological point of view

- *Case example:* Demonstrating a relevant exemplary case with the prototype.
- *Formal proof:* Theorem proving via a sequence of mathematical steps when some predefined statements (axioms) are assumed to be true.

1.7. Research strategy

Ontological, epistemological and methodological paradigms shape the research strategy and they have been applied for scientific research in different disciplines. Research strategy is an overall strategy based on the research philosophy for conceptualizing and conducting an inquiry, and constructing scientific knowledge [Hol97]. Furthermore, general research strategy patterns or research frameworks can help to describe the structure of the research.

Based on the research questions and the research philosophy explained in Section 1.5 and 1.6 respectively, this research has the characteristics of a deductive research while proposing a design artifact to improve the existing modeling and simulation methodologies. Hence, we have chosen the traditional scientific method for this research which is illustrated in Figure 1.4 [Cre03, HGG03, CJT10]. The steps of the research process are described as below:

1. *Identify the issues and ask research questions:* After choosing the research domain and the research topic, the problems or issues are identified. The objective of the research is stated and the research questions are defined to address the selected issues. In this research, our research domain is modeling and simulation, and our focus is on M&S methodologies, M&S lifecycle and simulation conceptual modeling. So, this thesis mainly contributes to the Body of Knowledge (BoK) of M&S Science [Tol13].

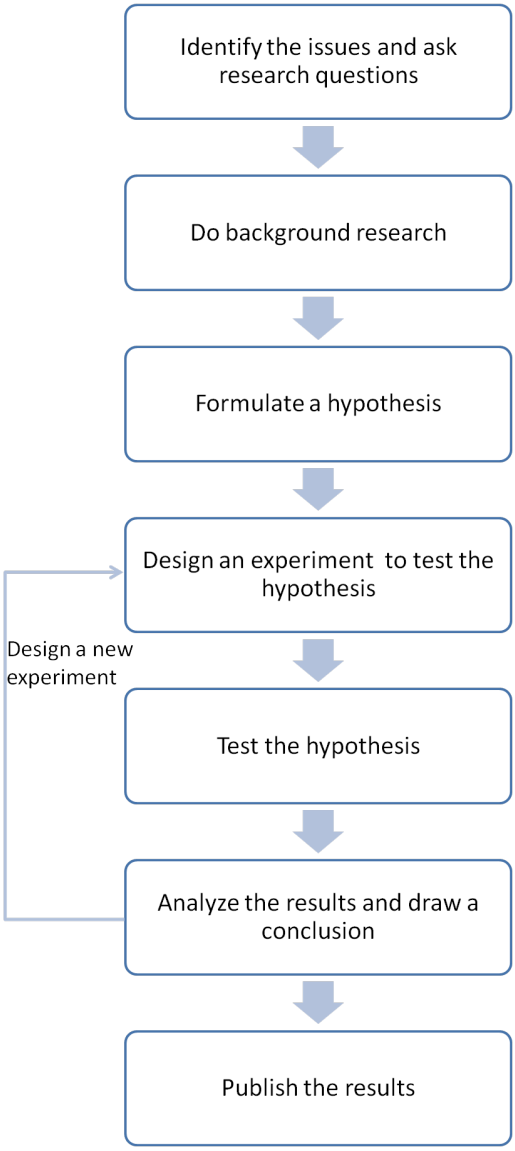


Figure 1.4: Research strategy.

2. *Do background research:* During the background research, existing knowledge about the research topic is reviewed. Different methods can be used to gather data and background information. In this research, mostly literature survey is used.

3. *Formulate a hypothesis:* The research hypothesis is formulated based on the background information.

4. *Design an experiment:* After formulating the hypothesis, an experiment to test the hypothesis is designed. In this research, first an MDD framework for M&S is proposed and a proof of concept implementation is developed to do experiments with the framework. Then, a case example is designed to test the hypothesis.

5. *Test the hypothesis:* The hypothesis is tested by performing the experiment. In this research, the designed case example is performed to test the positive effects of MDD on the simulation model development process.

6. *Analyze the results and draw a conclusion:* The results are analyzed to see if they support the initial hypothesis and conclusions are drawn.

7. *Publish the results:* Finally, the outcome of the research study is published and communicated with others. It is also important to present the partial results throughout the research in the well known workshops, conferences or journals for scientific review.

1.8. Outline of the thesis

The outline of this research is shown in Figure 1.5. The thesis is organized as follows:

Chapter 1 presents the motivation for this research and explains the research approach. Chapter 2 presents the background information. Section 2.1 focuses on simulation conceptual modeling. A set of M&S methodologies is analyzed in order to see if an MDD approach can be incorporated into these methodologies. Section 2.3 introduces model driven development into simulation field by explaining the principles of modeling, metamodeling and model transformations.

Chapter 3 proposes the MDD4MS framework as a new method for bridging the gap between the models at different steps of the M&S lifecycle and supporting model continuity. Chapter 4 explains how to use domain specific languages with the proposed framework. Chapter 5 explains how to utilize component based simulation to support model transformations. The outcome of the Chapters 3, 4, and 5 forms the theoretical underpinnings of this research.

Chapter 6 presents a proof of concept implementation of the MDD4MS framework in the business process modeling and discrete event simulation domains. Two case

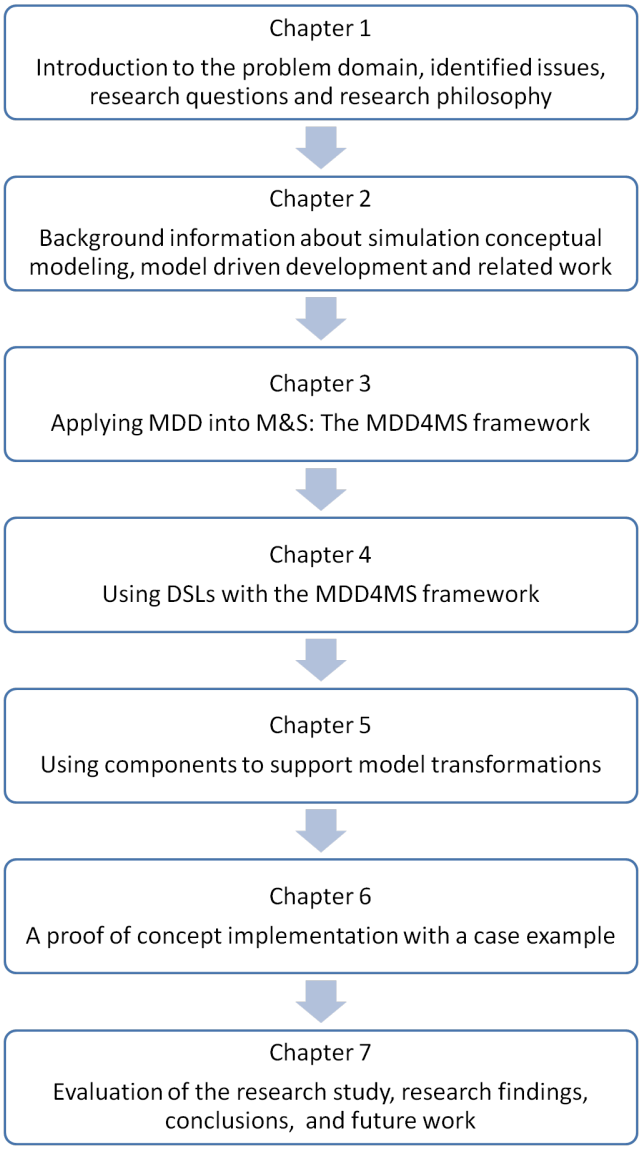


Figure 1.5: Research outline.

examples are presented to illustrate the usage of the framework and the prototype modeling environment. The results are analyzed to see if the prototype and the examples support our hypothesis. Finally, Chapter 7 presents the evaluation of the research study, draws the conclusions and suggests the future work.

1.9. Declaration

The content of this thesis is the result of the authors original work, except where referenced or stated otherwise. Parts of this dissertation have been previously published by the author. The following publications [ÇVS10a, ÇVS10b, ÇVS10, ÇVS11, ÇV11, RÇSW11, ÇVS12] are presented at international conferences. The following two chapters [ÇMVS13, ÇVS13] are published in a book, and the following article [ÇVS14] is submitted to a journal.

Chapter 2

Background and Related Work

In the previous chapter, a number of research questions are formulated to address the selected issues in the M&S field. This chapter provides background information to answer these research questions. Due to the fact that similar issues are identified in the software engineering field in the past and solved with model driven development approaches, this research proposes the application of a model driven development approach into the whole M&S lifecycle.

Firstly, a brief introduction to simulation conceptual modeling is given and the requirements for an effective conceptual modeling stage are presented in the next section. Then, a set of M&S methodologies are analyzed in Section 2.2 in order to show the lack of guidance for formal model transformations, as well as to show the possibility of embedding an MDD approach into these methodologies. After that, a detailed explanation of the MDD principles is provided with clear examples in Section 2.3. Finally, the related work about applying MDD into M&S is presented in Section 2.4.

2.1. Conceptual modeling for simulation

In general, conceptual modeling is a process that elicits the general knowledge about a problem/research domain and describes a conceptual model which is necessary to develop a solution for a given problem or a construct for a specific purpose [Oli07]. In a similar way, it can be said that simulation conceptual modeling is about moving from an M&S study definition and system requirements to a simulation conceptual model which shows the general knowledge about the problem domain and what is going to be developed in the simulation model [Oli07, RBKvdZ10]. A simulation conceptual model is an abstract representation of a system that describes the elements, relationships, boundaries and assumptions without reference to the specific implementation details [Fis95, Pac00, Rob06, Rob08a]. In this way, it is expected to show the structure and the abstract behavior of a system according to the purpose of the simulation study.

The perspective of the client and the modeler are both important in conceptual modeling [RBKvdZ10]. A conceptual model provides a means of communication

between all parties in a project and minimizes the likelihood of incomplete and inconsistent results. It is clear that without a good and agreed conceptual model, an excessively high share of the simulation project success responsibility is put in the hands of the simulation model programmer. This situation would have been mitigated only if stakeholders were involved in the conceptual modeling stage, and if the conceptual model is used explicitly in the further stages.

Although the term includes the word 'conceptual', a conceptual model is a concrete model which expresses the mental model in the mind of the conceptual modeler. A conceptual model can be a user's model or a design model, which can be compared to as-is and to-be models in business process modeling. Hence, it is specified in a variety of communicative forms such as chart, diagram, drawing, equation, graph, image, text, animation, audio or video [BO07, vdZKT+10]. The targeted users of a conceptual model can include project managers, analysts, designers, developers or simulation model programmers [vdZKT+10]. Therefore, it is very important to utilize a common language so that conceptual models are represented and communicated in a manner that is understandable to all users.

A conceptual model itself is a non-executable model since it is specified to be a preliminary model which will be used as a base for simulation model development [Nan84, Fis95]. So, it cannot be simulated directly. However, a conceptual model needs to be transformed into a simulation model by using some transformation rules, techniques or patterns. If a transformation method is formally defined and automatized, then a conceptual model can be transformed into an executable simulation model automatically based on these specific transformation rules, techniques or patterns.

The research on simulation conceptual modeling has increased in the last decade since conceptual modeling still remains a process that is almost completely performed casually. Zhou et al.[ZSC04] identify and address the issues in developing efficient models to capture, represent and organize the knowledge for developing conceptual models. Van der Zee et al. [vdZvdV07] discuss guidance offered by diagramming techniques and simulation tools during conceptual model development.

Robinson [Rob08b] presents a conceptual modeling framework that consists of the following activities: understanding the problem situation, determining the modeling and general project objectives, identifying the model outputs (responses), identifying the model inputs (experimental factors), determining the model content (scope and level of detail), identifying any assumptions and simplifications. Kotiadis and Robinson [KR08] recommend the use soft systems methodology [Che99] in undertaking knowledge acquisition and model abstraction during conceptual modeling and they provide examples in discrete event simulation.

Birta and Arbez [BA07] present another conceptual modeling framework for discrete-event dynamic systems. The concepts underlying their approach are based on activity scanning world view. An improved version is represented in [AB10]. Tako

et al. [TKV10] describe a framework and tools that enable stakeholder participation in the development of conceptual models in simulation studies. The suggested framework utilizes tools from soft systems methodology and group model building in system dynamics. A report of NATO Modeling and simulation group [NAT12] provides comprehensive information about conceptual modeling for military modeling and simulation. In the report, various methods are analyzed as well. Chwif et al. [CBdMFS12] present a framework for specifying discrete event simulation conceptual models. The framework extends the modeling framework of Robinson given in [Rob08b].

Looking at the mentioned frameworks, conceptual modeling has two sub-stages which may be performed in parallel: system structure definition and abstract behavior definition. Hence, the conceptual modeling method should support these activities for a complete conceptual model [KWHL03]. The commonly used approach to provide an accurate model is combining different diagramming techniques.

UML class diagrams, entity-relationship diagrams and system entity structure (SES) are used commonly for representing system structure. As well as ontologies provide a substantive basis for such a system structure definition in a domain [MA09]. A number of researchers have suggested using ontologies to propose simulation conceptual modeling methods. For example, Miller et al. [MBSF04] investigate the benefits of ontologies for discrete-event simulation and presents a Discrete-event Modeling Ontology (DeMO). Silver et al. [SHM07] present an ontology driven simulation method for using ontologies during the development of simulation models. It suggests a technique that establishes relationships between domain ontologies and a modeling ontology and then uses the relationships to instantiate a simulation model as ontology instances.

On the other hand, various modeling languages provide an effective way for abstract behavior definition. Simulation conceptual modeling mostly benefits from general purpose modeling languages or diagramming techniques, such as process flow diagrams, flowcharts, event graphs, activity cycle diagrams, IDEF diagrams, UML activity diagrams, etc. [SCT03, Ong10]. As well as, there are some domain specific conceptual modeling methods such as BPMN [OMG11a], DEMO methodology [Die06], KAMA [BD08], Simulation Modeling Language (SimML) [Ari01], Simulation Model Definition Language (SMDL) [ESA05], Simulation Reference Markup Language (SRML) [Rei02] and OntoUML [GW12].

Higher level visual diagrams for some formalisms such as Petri Nets [Pet81] or DEVS [ZPK00] are used for conceptual modeling as well. Although these formalisms are not conceptual modeling languages, the higher level incomplete models can still be used as a conceptual model if all of the stakeholders are familiar with the formalism. In this case, conceptual models are refined incrementally and the final full model becomes the simulation model. But, in most cases, problem owners are not familiar with the chosen system specification formalism and so it is hard to communicate during the conceptual modeling stage. Hence, it is critical to

use a conceptual modeling language such that both the problem owner and the simulation modeler can understand the conceptual model, and they can agree on it. In general, a conceptual modeling language has no execution semantics while a system definition language has execution semantics. Model continuity problem arises when there is no formal continuity relation between the models when different modeling languages are used at each stage. Although there are many languages, techniques and tools for simulation conceptual modeling, more formal and precise methods and tools are needed that will enable the explicit use of the conceptual models [Pac00].

2.1.1. Requirements for simulation conceptual modeling

This section presents key requirements for conceptual modeling to provide a sound reference for simulation model development. These requirements can form a basis for an effective conceptual modeling method for simulation. During the conceptual modeling stage:

- **R-CM.1.** The problem/research domain ontology must be described [GW04],
- **R-CM.2.** A modeling language must be chosen to specify the conceptual model [GW12],
- **R-CM.3.** The conceptual model must conform to the modeling language,
- **R-CM.4.** The system structure and abstract behavior must be defined [Rob08a],
- **R-CM.5.** The boundaries and the assumptions must be defined [Rob08a],
- **R-CM.6.** The conceptual model must be communicative between the stakeholders [vdZKT+10],
- **R-CM.7.** The conceptual model must be independent from the implementation details [Rob08a].

During conceptual modeling stage these requirements need to be satisfied. Choosing a suitable and effective conceptual modeling method can help to develop a good conceptual model. Domain specific modeling languages can help by defining the concepts for a specific domain. Metamodeling approach can be utilized to define both ontologies and modeling languages in an efficient and well-defined way. The MDD4MS framework proposed in Chapter 3 includes well-defined solutions for conceptual model representation via metamodeling and Chapter 4 provides more information about domain specific languages.

2.1.2. What is next? From conceptual models to simulation models

Regardless the problems of simulation conceptual modeling in practice, if we assume that a successful conceptual modeling stage is performed and a good conceptual model is developed by using a suitable conceptual modeling language, there are still problems while moving to the next stage in the project lifecycle. Conceptual models are often not used explicitly in the further stages as formal model transformation methods are not available to provide model continuity. Although a good conceptual model increases the quality of a project and provides a good understanding of the problem, a high share of the responsibility is again put in the hands of the simulation model programmer if there is no explicit model continuity.

As a result, the existing M&S methodologies and conceptual modeling methods require some development in the state-of-the-art approaches. This research proposes the use of the model driven development methods, techniques and tools to improve the M&S activities. To introduce the MDD approach into the simulation field, a number of M&S methodologies are analyzed according to their support for MDD. The major requirement for an MDD supporting methodology is defining a multi-stage and multi-model development lifecycle [Sel06].

2.2. Analysis of M&S methodologies for applying MDD

In this section, a set of M&S methodologies will be analyzed in order to see if an MDD approach can be incorporated into these methodologies [ÇVS14]. Each methodology is evaluated according to a set of questions to analyze the support for model continuity. The questions are defined according to the required steps in a simulation study. Table 2.1 presents the results and it is shown that all of the methodologies introduce multiple stages for carrying out a simulation study. Regarding the outputs of the sub-stages it is identified that the development process highly relies on models at different abstraction levels [TDT08].

All of the methodologies require a conceptualization step and many of them suggest to transform the conceptual model into a simulation model, but do not formally define how to do that. All of the methodologies prepare for computer simulation and require a programming or model coding step. Some of them mention automatic code generation. Therefore, M&S can be identified as a model based process [Ö07] and an MDD method can be incorporated into these methodologies. On the other hand, some of the methodologies are lacking a formal specification step which is important to separate the formal system specification and programming concerns. Moreover, almost all of the methodologies ignore model transformations and none of them provides formal model transformation methods between the models at different abstraction levels. Hence, model continuity is not supported in many cases. Due to the similarities with software engineering, we believe that MDD principles can be applied to M&S and existing MDD tools and techniques can be utilized. The next section explains the principles of MDD.

Table 2.1: Analysis of the M&S methodologies according to their support for model continuity [ÇVS14].

Analysis criteria	[Sha75] p.24	[RAD+83] p.8	[Nan84] p.76	[Fis95] p.4	[Sha98] p.9	[Ban98] p.15	[Law03] p.67	[Rob04] p.52	[Wai09] p.27	[vDTVB09] p.1129	[Sar10] p.169	[Bal12] p.4
Has multiple stages?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Has multiple models?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
How many models?	2	3	4	6	2	2	2	2	3	3	2	2
Has a conceptualization step?	P[1]	P[1]	Y	Y	P[1]	Y	Y	Y	Y	Y	Y	Y
Has a formal specification step?	N	Y	N	Y	N	N	N	P[3]	Y	Y	P[3]	P[3]
Has a programming step?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Suggests conceptual model transformation?	N	P	Y	Y	P	Y	N	Y	N	Y	N	Y
Suggests automatic code generation?	N	N	Y	Y	N	Y	N	N	N	N	N	Y
Provides formal model transformations?	N	N	N	P[2]	N	N	N	N	N	N	N	N

Note: Y: Yes, N: No, P: Partially.

[1] Conceptual model is not mentioned, but a diagram such as a causal-loop diagram or a flow diagram is suggested.

[2] The framework does not provide formal transformation methods, but suggests some heuristics, explicit translation rules and general code writing rules.

[3] A software specific description that determines how to structure the model in the chosen software is suggested.

2.3. What is MDD?

MDD is a software development methodology that provides a set of methods and guidelines to develop a software system through successive model transformations [ÇV11]. In MDD, models are the primary artifacts of the development process and they represent the system and the software at different levels of abstraction or detail. The models are transformed into other models at different stages in order to (semi)automatically generate the final software system. MDD produces well-structured and maintainable systems. The most important MDD methods are modeling, metamodeling and model transformations.

Modeling is the process of representing a source system for a specific purpose in a form that is ultimately useful for an interpreter [ÇVS14]. The concrete form that represents the system is called the model. A model is specified in a modeling language. The MDD approach requires that the models and modeling languages are well defined. Metamodeling is the most commonly used method to describe a modeling language formally in the form of a metamodel, which in turn can be used to specify models in that language. Model transformation is the process of converting a model into another form according to a set of transformation rules. Model transformations are performed to utilize the knowledge in an existing model.

In an MDD approach, usually there is a chain of several model transformations and a final model-to-code transformation. The approach is based on the idea of having several intermediate models. Each model represents a different view of the system. Although MDD has been advocated as a cost and effort saving development approach for software projects [Sel03], the MDD principles are currently only described informally. Information about the conceptual application of MDD principles are provided by different specifications such as Model Driven Architecture (MDA) [OMG03], Model Integrated Computing (MIC) [ISI97], Eclipse Modeling Framework [Ecl09], and Microsoft Software Factories [Mic05]. These specifications explain the steps required to take a model from conceptual design to final implementation. However, the software engineering community does not have a sound and complete theory for MDD [Fav04].

MDA is a software design and development approach that provides a set of guidelines for specifying and structuring models. MDA prescribes the use of metamodels and meta-metamodels for specifying the modeling languages without any necessity to be domain specific. Object Constraint Language (OCL) can be used for defining constraints over metamodels as well as actual models in order to precisely define a modeling language. Models can be exchanged in XML Metadata Interchange (XMI) format.

MIC refines the MDD approaches and provides an open integration framework to support formal analysis tools, verification techniques, and model transformations in the development process. MIC allows the synthesis of application programs from models by using customized model integrated program synthesis (MIPS) envir-

onments. The meta level of MIC provides metamodeling languages, metamodels, metamodeling environments, and meta generators for creating domain specific tool chains on the MIPS level. The fully integrated meta programmable MIC tool suite provides open-source tools.

Microsoft Software Factories is a unified software development approach, which tries to synthesize ideas from DSLs, MDD, software product lines and development by component assembly. When compared to MDA, Software Factories are less concerned with portability and platform independence and more concerned with productivity.

The Eclipse Modeling Project focuses on the evolution and promotion of the MDD technologies within the Eclipse community. It provides a unified set of modeling frameworks and open source tools. The following projects are all included in Eclipse Modeling Project: EMF, generative modeling technologies, M2M transformation, M2T transformation and model development tools.

Due to the fact that different MDD tools apply the same principles, an MDD expert can easily think about combining different approaches and tools to perform an MDD process. For example, the MDA concepts can be implemented with the MIC tool suite. In MDD literature, the most commonly used and accepted specification is MDA [OMG03]. The MDA concepts are presented in terms of some existing or planned system, where a system mainly refers to a software application within a system.

MDA introduces three types of viewpoints: The computation independent viewpoint focuses on the environment of the system, and the requirements for the system. The platform independent viewpoint focuses on the structure and operation of a system while hiding the details necessary for a particular platform. The platform specific viewpoint focuses on the use of a specific platform by a system. Based on these viewpoints, three types of models are used in MDA: A Computation Independent Model (CIM) is a representation of a system from the computation independent viewpoint that does not show the details of the structure of the system. A Platform Independent Model (PIM) is a representation of a system that exhibits a specified degree of platform independence to be usable with a number of different platforms. A Platform Specific Model (PSM) is a representation of a system that combines the specifications in the PIM with the details that specify how that system uses a particular type of platform. MDA also defines a model transformation as the process of converting one model to another model of the same system. It explains the PIM-to-PSM transformation and requires that the PSM will include the source code if it is an implementation. However, CIM-to-PIM transformation is not clearly defined in MDA since a CIM is assumed to be a kind of requirements specification.

By following the MDA viewpoints, we can say that at least three types of models are produced during a software development lifecycle: a CIM for the analysis stage,

a PIM for the design stage, and a PSM for the implementation stage [Som07]. The final source code will be generated from the PSM. In more generic terms, an MDD process is supposed to have an initial model, a number of intermediate models and a final source code. The aim is to obtain a large part of the intermediate models and the final code through successive model transformations. An MDD process supports model continuity via formal model transformations.

The most notable advantages of MDD are rapid software development and increased productivity. Although the advantages of quality, accuracy, maintenance, customer satisfaction, support for documentation, validation and verification are commonly accepted as pros of MDD, it is often examined for being really faster than traditional development [MD08, MCM13]. The answer depends on the tools that are being used, the abilities of the team, the domain knowledge of the metamodel developers, the availability of the suitable DSMLs, expected software quality and so on. When the modeling languages are not available and the team members have little or no knowledge about MDD, it may take a large amount of time to develop metamodels. However, once they are developed then the further development time and costs decrease significantly.

MDD is different from the traditional development approaches and it requires a learning period and changing the programming habits. However, working with meta-levels is easier and beneficial as soon as it is understood well. Although traditional systems modeling and software engineering approaches can be chosen in small-scale and short-term projects, the model driven approach is more desirable for large-scale and critical projects. The following sections provide more information about modeling languages, metamodeling and model transformations.

2.3.1. Modeling languages

A modeling language is a means of expressing systems in a formal and precise way by using diagrams, rules, symbols, signs, letters, numerals, etc. A modeling language consists of the abstract syntax, concrete syntax and semantics [AK03, FB05]. The abstract syntax describes the vocabulary of the concepts provided by the modeling language and how they can be connected to create models. The abstract syntax consists of the concepts, the relationships and well-formedness rules, where well-formedness rules state how the concepts may be combined. The concrete syntax provides a way to show the modeling elements in a concrete form which we see and work with on paper or on the computer screen [Rum98]. The semantics of a modeling language is the additional information to explain what the abstract syntax actually means.

Computerized tool support is very important in MDD approaches. In order to specify, view or change models on computer platforms, we use model editors. A model editor for a modeling language l provides a way to specify models according to the syntax of l and save them according to a specific file format. Besides, the editor uses a language parser to decompose a model according to the abstract

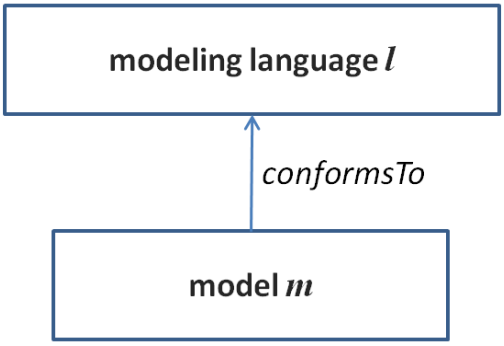


Figure 2.1: A model conforms to a modeling language.

syntax of l [MFF+08] and shows the model on the screen using one or more views. In many cases, the model editor provides extra features such as verification or syntax highlighting.

The relation between a modeling language and a model expressed in that language is called the *conformsTo* relation, such as "the model m *conformsTo* the modeling language l". *conformsTo* relation shows that model is specified according to the modeling language specification. Model verification ensures that this relation is correct. Figure 2.1 illustrates the *conformsTo* relation between a model and a modeling language.

Example: Simple state diagram modeling language

Figure 2.2 illustrates an example of a simple state diagram modeling language based on the formal semantics of a Mealy machine. A state diagram is used to represent the states and state transitions of a system. State means all the stored information about the system properties at a given point in time. A state diagram is a directed graph with states as vertices and transitions as edges. Each state in a diagram refers to a set of predefined values of the system properties. Inputs are used either to change the system properties or to produce an output at a certain state. A state diagram d for a finite state machine is a directed graph, $d = (S, s_0, X, Y, \delta, \lambda)$ where,

S is a finite set of states (vertices)

s_0 is an initial state

X is a finite set of input symbols

Y is a finite set of output symbols

a) abstract syntax

```
<diagram>      ::= <states> <transition>*
<states>       ::= <initialState> <state>*
<initialState> ::= <state>
<state>        ::= "state[" <stateID> ", " <name> "]"
<transition>   ::= "conn[" <stateID> ", " <stateID> ", "
                                  <input> ", " <output> "]"
<stateID> ::= <string>
<name>    ::= <string>
<input>   ::= <integer>
<output>  ::= <integer>
```

b) concrete syntax

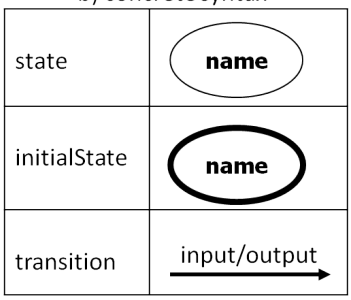


Figure 2.2: Syntax of a simple state diagram modeling language.

$\delta : S \times X \to S$ is the transition function (edges)

$\lambda : S \times X \to Y$ is the output function

In Figure 2.2, <states> is the finite set of states defined by an ID and name. <initialState> is the initial state and <transition> is a connection from a state to another state defined by $\delta(input)$. The output defined by $\lambda(input)$ is written together with the transition connection. Figure 2.3 shows an example model with the given state diagram modeling language.

2.3.2. Metamodeling

Metamodeling, in MDD context, is the process of complete and precise specification of a modeling language in the form of a metamodel. The metamodeling term has been used in simulation for many years in a different context. Metamodeling referred to constructing simplified models for simulation models that are quite complex [Bar98, Kle09]. In this context, a metamodel is known as a surrogate model. Surrogate models mimic the complex behavior of the underlying simulation model. Metamodeling in MDD context has been introduced to simulation lately.

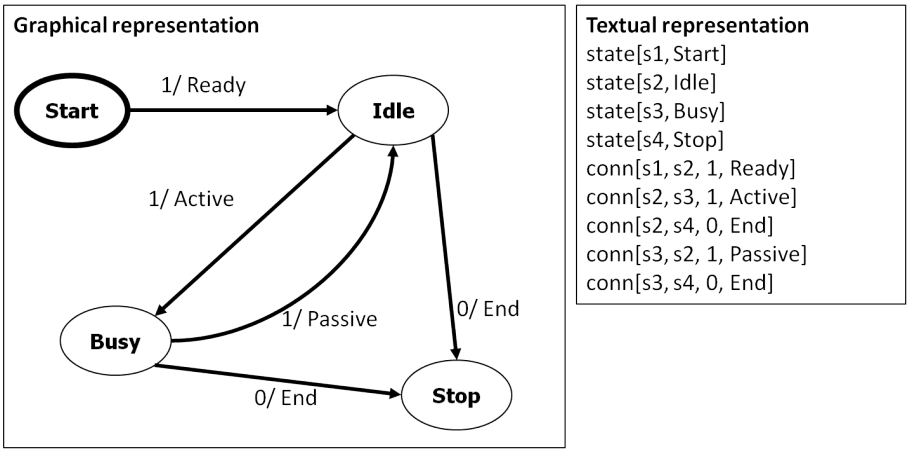


Figure 2.3: A model with the example state diagram modeling language.

The purpose of the metamodeling process is representing the abstract syntax of a modeling language with a metamodel in order to provide a proper way to develop models with that language. For example, instead of developing a model for a specific problem in a certain domain, first a metamodel which defines the concepts that apply to a larger set of problems in that domain is specified. Then, the metamodel is used to develop a specific model. In this case, a model is said to be an instance-of the metamodel. Hence, in a metamodeling approach, the *conformsTo* relation between the model and the modeling language is implicitly expressed via an *instanceOf* relation, such as "the model is an *instanceOf* the metamodel". While the *conformsTo* relation only guarantees a valid model, the *instanceOf* relation requires that every element of a model must be an instance of some element in the metamodel.

A metamodel is specified in a metamodeling language. Similar to a modeling language, a metamodeling language has one abstract syntax and at least one concrete syntax. Figure 2.4 illustrates the relations between a model and a metamodel. The *conformsTo* relation between the model and the modeling language is shown with a dashed line since this relation is implicitly obtained. One can easily notice that the metamodeling pattern can be applied again as a metamodeling language can also be defined with a metamodel. In this case, the metamodel, which is a model of the grammar of a metamodeling language, is called a meta-metamodel and it is specified in a meta-metamodeling language. To summarize,

- A model represents a system.
- A modeling language is used to specify models.
- A metamodel represents a modeling language.

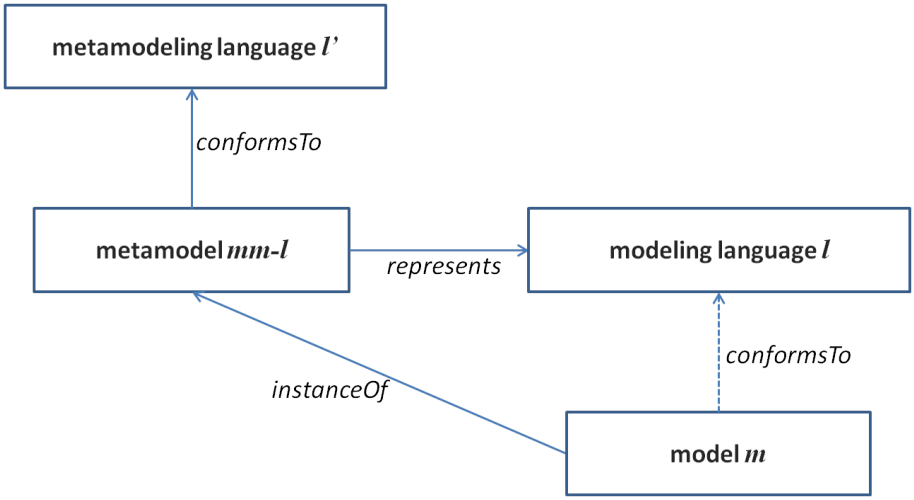


Figure 2.4: Metamodeling.

- A metamodeling language is used to specify metamodels.
- A meta-metamodel represents a metamodeling language.

Due to the fact that both a metamodel and a meta-metamodel are used to represent languages, meta-metamodels are often self describing to avoid an unnecessary stack in the number of meta levels. Therefore, the meta-metamodel conforms to the metamodeling language that it represents. Most approaches implement a three level metamodeling stack for model, metamodel and meta-metamodel in practice. Figure 2.5 illustrates the relations between a metamodel and a meta-metamodel.

Although, it is possible to increase the number of metamodeling levels theoretically, a 4-level metamodeling architecture that was introduced in the UML specification by OMG in 1999 [OMG99] is generally used in practice. The classical UML specification is based on four levels, but the later versions allow more or less meta-levels than this [OMG11c]. However, the minimum number of levels is two. In the metamodeling architecture introduced by OMG, the M3-level is for representing a metamodeling language as a meta-metamodel; the M2-level is for representing a modeling language as a metamodel; the M1-level is for representing a system without specific user data; and the M0-level is for representing a system with user data. Table 2.2 shows the metamodeling levels introduced by OMG. The Meta-Object Facility (MOF) specification is the defacto metamodeling language in OMG specifications for the M3 level where models can be created, integrated and transformed into different formats [OMG06].

In addition, OMG introduces the very useful concept of UML profile at the M1-level.

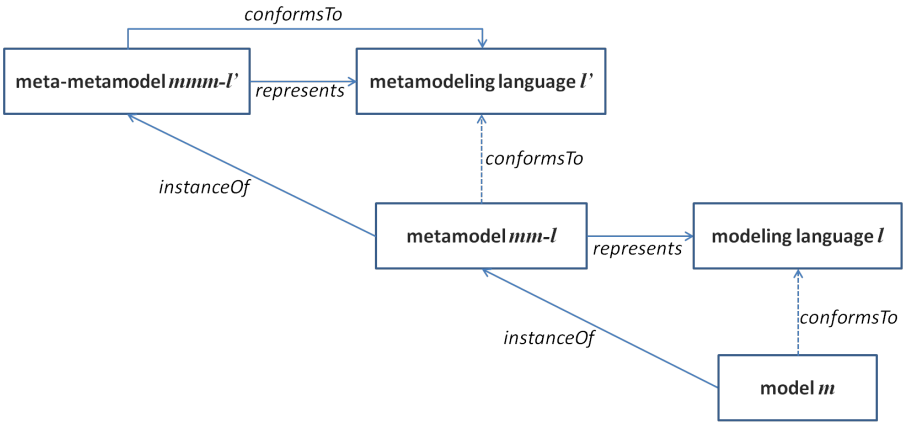


Figure 2.5: Self describing meta-metamodel.

It is a way of specifying an incomplete (parameterizable) model, so that the details can be filled in later. We will use the term *template models* for these incomplete models within the modeling process. A set of template models in a modeling language for a domain is called a domain-specific modeling library, whereas the set of modeling elements provided by the grammar of the language is called the basic (or core) modeling library [AK02]. We accept these features as implementation specific concepts, and many state-of-the-art metamodel editors provide them.

2.3.3. Metamodeling languages

Popular metamodeling languages are: Meta-Object Facility (MOF) [OMG06], Ecore [Ecl09], KM3 [IRI11] and MetaGME [ESB04]. OMG's MDA relies on the MOF to integrate the modeling steps and provide the model transformations [OMG03]. The earlier version of the MOF meta-metamodel is a part of the ISO/IEC 19502:2005 standard [ISO05]. The MOF meta-metamodel is referred as the MOF Model and it is self-describing, i.e. it is formally defined using its own metamodeling constructs. In 2006, OMG introduced the Essential MOF (EMOF) which is a subset of MOF with simplified classes [OMG06]. The primary goal of EMOF is to allow defining metamodels by using simpler concepts while supporting extensions.

Eclipse Ecore meta-metamodel is the main part of The Eclipse Modeling Framework (EMF) project. The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Ecore is based on the EMOF specification, but renames the metamodeling constructs like EClass, EAttribute, EReference or EDataType. An Ecore metamodel needs to have a root object and a tree structure represents the whole model.

MetaGME is the top level meta-metamodel of the MIC technology. MIC defines a

Table 2.2: Four layer metamodeling architecture in UML specification [OMG99].

Levels	Description	Example
M3: meta-metamodel	Defines the language for specifying metamodels.	MetaClass and MetaAttribute (in the MOF Specification)
M2: metamodel	Defines the language for specifying models. An instance of the meta-metamodel.	Class and Attribute (in the UML Specification)
M1: model	Defines the model without user data. An instance of the metamodel.	Car class and Name attribute (in a template UML Model)
M0: runtime model	Defines the model with user data. A runtime instance of the model.	Car123 with Name="abc" (in a certain UML Model)

technology for the specification and use of the DSMLs. MIC refines and facilitates MDD approaches and provides an open integration framework to support formal analysis tools, verification techniques and model transformations in the development process. MIC allows the synthesis of application programs from models by using customized Model Integrated Program Synthesis (MIPS) environments.

Kermeta is a metamodeling language which allows describing both the structure and the behavior of models. Kermeta is built as an extension to EMOF and it provides an action language for specifying the behavior of models. Kermeta is fully integrated with Eclipse and it is available under the open source Eclipse Public License (EPL).

2.3.4. Metamodeling tools

In order to specify, view or change metamodels on computer platforms, metamodel editors are used. A metamodel editor uses a language parser to decompose a given metamodel according to the abstract syntax of the chosen metamodeling language. In general, the metamodel editors provide extra features such as model-to-model transformation, code generation or model interpretation. In this case, the complete tool set is called a metamodeling environment. A full-featured metamodeling environment provides a way to specify a metamodel mm and automatically generate a model editor for the modeling language that mm represents. The resulting editor may either work within the metamodeling environment, or less commonly be produced as a separate standalone program.

Some well known metamodeling environments are Generic Modeling Environment

(GME) [ESB04], MetaEdit+, AndroMDA, Microsoft's DSL Tools for Software Factories [Mic05], ASF+SDF Meta environment [vdBvDH+01], AToM3 [dLV02], XMF-Mosaic, the Eclipse generative modeling technologies (GMT) project [Ecl06], and Kermeta development environment [IRI11]. Metamodeling had always a close relationship with Domain Specific Languages (DSL) and Domain Specific Modeling (DSM). Hence, there have been some misuses of concepts such as metamodeling environments are generally called as DSM environments.

GME is a free and open source MIPS tool. In GME, metamodels are defined as modeling paradigms. Upon loading a modeling paradigm, the MetaGME interpreter automatically creates an environment for model development in the specified modeling language. GME has a decorator facility for nicer visualization of the models. It is also possible to define some constraints on the metamodel with OCL.

Eclipse GMT project provides a set of prototypes and research tools in the area of MDD. Historically the most important operation was model transformation, but other model management facilities like model composition are also being proposed. Different sub-projects are developed in the GMT project. The Generic Eclipse Modeling System (GEMS) in GMT project is a configurable toolkit for creating domain-specific modeling and program synthesis environments for Eclipse. GEMS provides a visual metamodeling environment based on EMF and GEF/Draw2D. It includes a code generation framework that a graphical modeling editor is generated automatically from a visual metamodel specification. The graphical modeling editor can be used for editing instances of the modeling language described by the metamodel.

Kermeta development environment provides a comprehensive tool support for metamodeling activities, such as an interpreter, a debugger, a text editor with syntax highlighting and auto code completion, a graphical editor and various import/export transformations. Achilleos et al. [AGY07] and Amyot et al. [AFR06] present information about various metamodeling environments.

It is not easy to define a grammar and a concrete syntax for a modeling language textually from scratch. Besides, developing a model editor for this language requires software engineering skills. The metamodeling tools provide a precise way for defining metamodels and auto-generating model editors.

Example: A metamodel for the state diagram modeling language

Figure 2.6 illustrates a metamodel for the simple state diagram modeling language given in Section 2.3.1. The metamodel is specified with the GEMS plug-in in the Eclipse GMT project [Ecl06]. The auto generated modeling editor for the given metamodel is shown in Figure 2.7. Although the generated editor uses a default concrete syntax, the graphical representation of the modeling elements and the visual appearance of the editor can be customized by changing a style file.

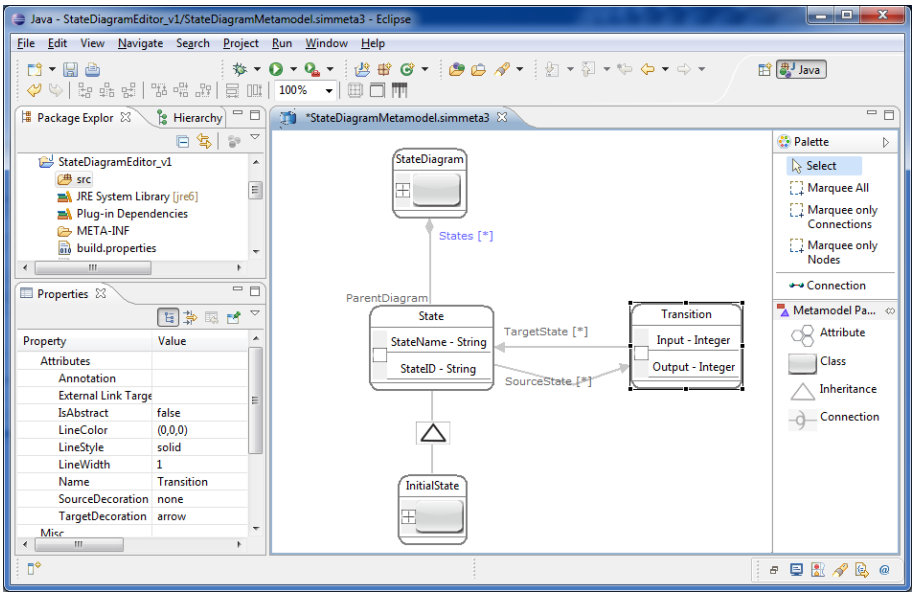


Figure 2.6: An example metamodel for the simple state diagram modeling language.

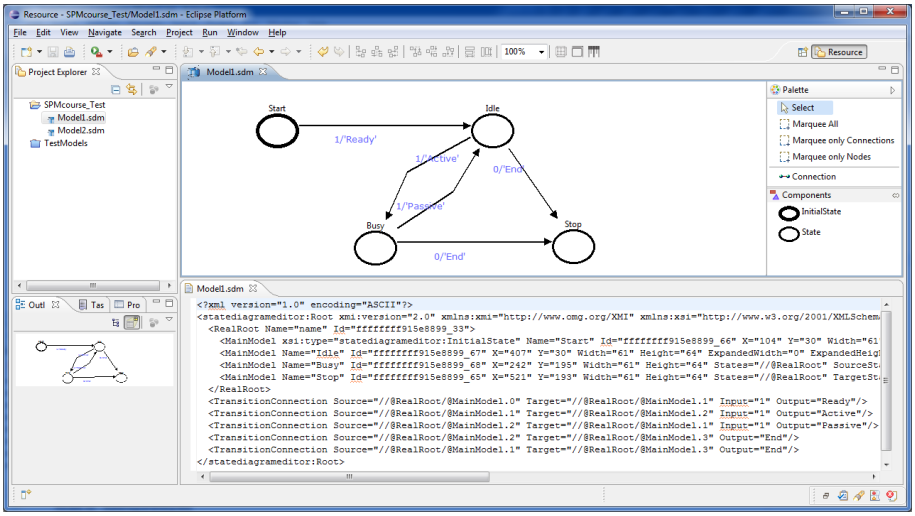


Figure 2.7: Auto generated modeling editor for the example metamodel.

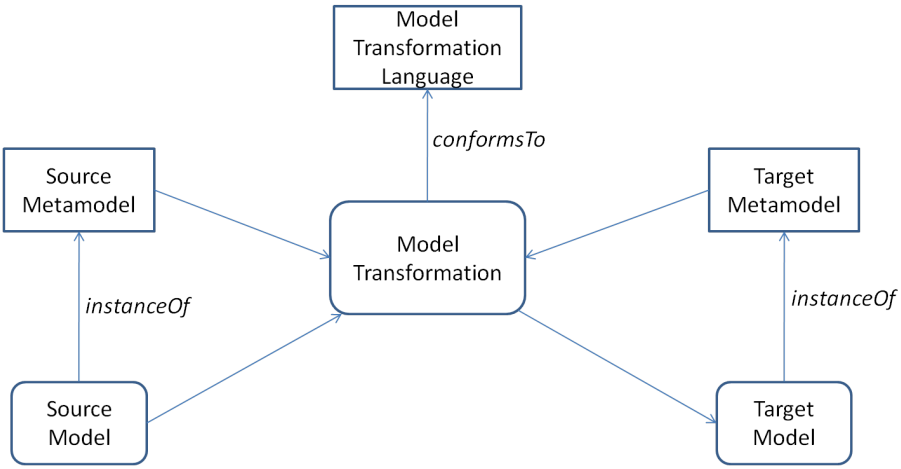


Figure 2.8: Model transformation pattern in MDD.

2.3.5. Model transformations

Instead of creating the models from scratch during the different development life cycle stages and activities, model transformations enable the reuse of information that was once modeled. A model transformation is the process of converting a model into another form according to a set of transformation rules. Model transformations are carried out to transfer the existing information in a model to a new model.

In the MDD context, we are only interested in formal model transformations. A formal transformation requires that the models are specified in well-defined modeling languages and the rules are defined with a model transformation language. A transformation rule consists of two parts: a left-hand side that accesses the given model; and a right-hand side that generates the target system. Hence, a model transformation is performed with a well-defined model transformation pattern. In order to provide model continuity, the target model should contain as much as possible from the source model and the initial modeling relation should be preserved [EE08]. Retaining the problem owner's language throughout the modeling process and keeping the concepts in the simulation model can lead to highly effective and successful projects [MR09]. During the transformation, the source model remains unchanged. Figure 2.8 shows a model transformation pattern, which is adapted from the MDA [OMG03].

According to the output type and the target language of the process, the model transformation can be classified as model to model (M2M) transformation, model to text (M2T) transformation, or model to code transformation (code generation). In an MDD approach, usually there is a chain of several M2M transformations

and a final code generation [CH03]. As the name implies, a M2M transformation converts a source model into a target model, which can be instances of the same or different metamodels [SVB+06]. However, a M2T transformation converts a source model into text. M2T transformation is generally used for final code generation or supportive document creation. If the model is used to generate source code, then the transformation is called code generation. For a metamodel based model transformation, either the source model is an instance of the source metamodel or the target model is an instance of the target metamodel, or preferably both.

If the source and target metamodels are identical, the transformation is called endogenous; otherwise it is called exogenous [MG05]. If the level of abstraction does not change, the transformation is called horizontal transformation. If the level of abstraction does change, the transformation is called vertical transformation. It is important to note that the possibility of defining various model transformations for the same source model. However, the common objective is to preserve and reuse the information in the source model as much as possible, which increases the percentage of the auto generated part in the target model, code, or text.

2.3.6. Model transformation languages

The goal of model transformations is to automatically generate different representations of a system at different abstraction levels and to enable the reuse of information that was once modeled. A key point here is the model transformation language. A model transformation language is a language that provides a way to write transformation rules for the expressions of a formal grammar. Given two formal grammars (one is for the source language and one is for the target language) and a model specified in the source language, a language parser can parse the transformation rules; an interpreter can interpret the source model; and a model transformation engine can apply a consecutive set of the rules on the source model and generate a target model according to the interpretation. A model transformation tool combines all of these concepts and it is usually embedded in a metamodeling environment. Sometimes, a model transformation is called a graph transformation if the models are specified as graphs, and the transformation tool is then called a graph transformation tool [ASK04]. A model transformation is also known as model morphism [ACJG10].

Well known M2M transformation languages are ATL (ATLAS Transformation Language), QVT (Query/View/Transformation), GReAT (Graph Rewriting and Transformation Language) and Xtend. MOF 2.0 QVT Specification is a set of model transformation languages defined by OMG [OMG11b]. The QVT specification has a hybrid declarative/imperative nature and it defines three related transformation languages: Relations, Operational Mappings, and Core. The Relations metamodel and language supports complex object pattern matching and object template creation with a high-level user-friendly approach. Operational Mappings can be used to implement one or more Relations from a Relations specification, when it is difficult to provide a purely declarative specification of how a Relation is to be

populated. The Core metamodel and language is defined using minimal extensions to EMOF and Object Constraint Language (OCL) on the lower level.

ATL is one of the most popular model to model transformation languages [BDJ+03, JABK08]. Once the target metamodel and source metamodel are available, an ATL transformation file can produce a target model from a source model. During the transformation, an attribute of a target model instance receives a return value of an OCL expression that is based on a source model instance. The ATL integrated development environment (IDE), which is developed on top of the Eclipse platform, provides a number of tools such as a text editor with syntax highlighting, a debugger and an interpreter that aims to ease development of ATL transformations.

There are many other model transformation languages [Sch06a]. As well as, some projects like Kermeta, Fujaba and GReAT (these projects apply an MDD approach) provide M2M transformation methods. Detailed explanation about the model transformation languages and methods can be found in [Hub08, CH03, DBAS09].

MDD provides a very generic approach such that everything can be a source of a modeling process, i.e. everything can be modeled. For example, it is possible to develop a model of a model transformation pattern [BBG+06]. Even further, a metamodel can be defined to represent the grammar of the model transformation language. In this way, it can be possible to define model transformation patterns graphically.

Example: Transforming state diagrams

In this section, the example model presented in Figure 2.7 will be transformed into a text file to illustrate a M2T transformation. The transformation is done by a visitor-based model interpreter whereas the mechanism is provided by the Eclipse GEMS project. Figure 2.9 shows a part of the model interpreter code in JAVA. For each modeling element specified in the metamodel, a corresponding visitor function is implemented. Then, the model interpreter is registered to the state diagram editor by adding an extension point to the plugin file. Figure 2.10 shows the generated output for the sample model.

2.3.7. Criteria for model transformations

As stated earlier, there are many ways to define a model transformation for a source model. Choosing the best model transformation is one of the challenging activities of MDD. There is no general rule or guidance that can be provided to define a good solution. However, both the target model and the transformation rules can be evaluated according to some basic principles derived from software engineering. This section provides evaluation criteria for model transformations extending the work of [MG05]. Based on these criteria, model transformations can be analyzed and results can be used to choose the best model transformation.

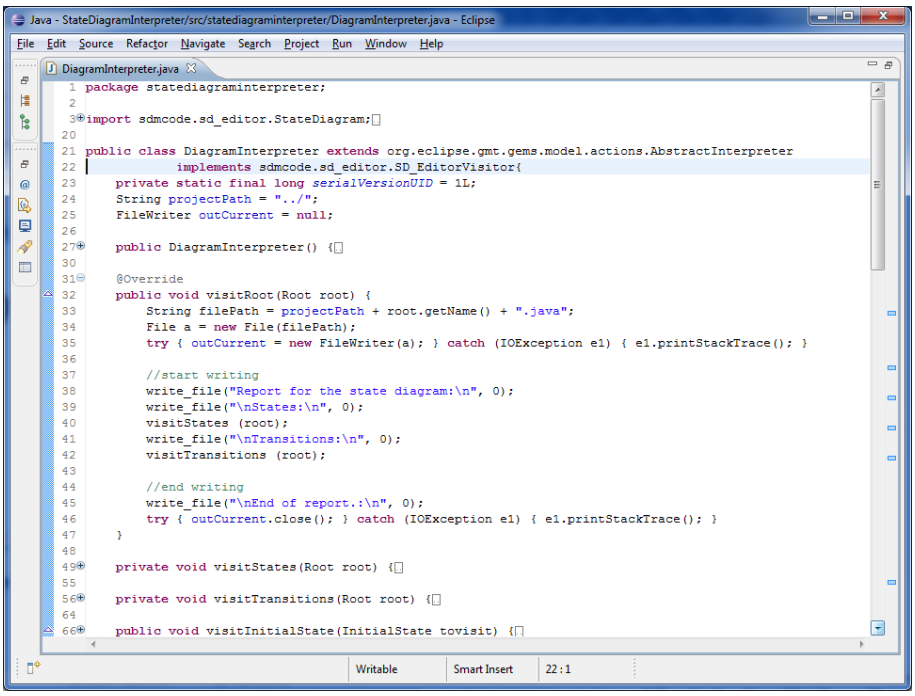


Figure 2.9: A visitor based model interpreter for a M2T transformation in JAVA.

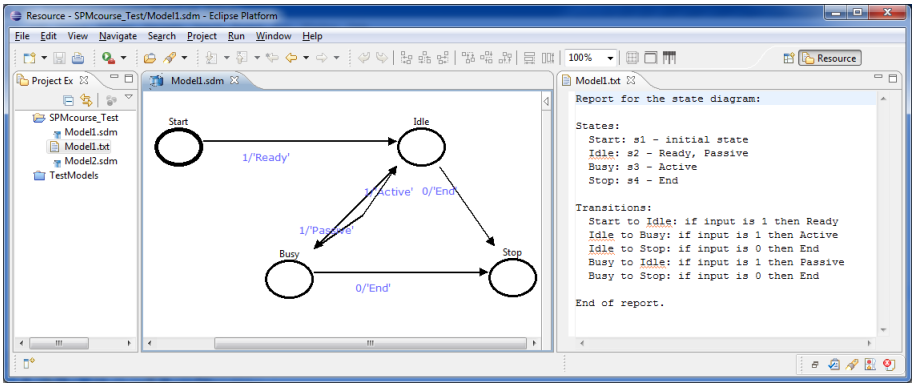


Figure 2.10: Output of the example model transformation.

- *Correctness*: The correctness of a model transformation is analyzed in two ways: syntactic and semantic correctness [MG05]. If the target model conforms to the target metamodel specification, then the model transformation is syntactically correct [EE08]. If the model transformation preserves the semantics of the source model, then it is semantically correct [NK08].
- *Completeness*: For each element in the source model, if there is a corresponding element in the target model then the model transformation is complete [MG05].
- *Termination*: If the model transformation always terminates and leads to a result, then it provides termination.
- *Uniqueness*: If the model transformation generates unique target models for each source model, then it provides uniqueness.
- *Readability*: If the transformation rules are human-readable and understandable, then the model transformation provides readability.
- *Efficiency*: The efficiency of a transformation can be evaluated by analyzing how many transformation steps are necessary, and how many functions are applied during each specific transformation.
- *Maintainability*: The degree of the effort spent for changing, extending and reapplying a model transformation defines the maintainability.
- *Scalability*: The ability to cope with large models without sacrificing performance defines the scalability [MG05].
- *Reusability*: Reusability can be measured with the possibility of adapting and reusing a model transformation via various reuse mechanisms such as parameterization or using templates [MG05].
- *Accuracy*: If all possible errors are handled and the model transformation can manage with the all incomplete source models, then it provides accuracy.
- *Robustness*: If most of the unexpected errors can be handled and the model transformation can manage with the all invalid source models, then it provides robustness.
- *Validity*: Since transformations can be considered as a special kind of software programs, systematic testing and validation techniques can be applied to transformations to ensure that they have the desired behavior [MG05].
- *Consistency*: If the model transformation detects and possibly resolves the internal contradictions and inconsistencies, then it is consistent [MG05].
- *Traceability*: Traceability is the property of having a record of links between the source and target elements as well as the various stages of the transformation process. Traceability links can be stored either in the target model or separately [DBAS09].

- *Reversibility*: A model transformation from s to t is reversible if there is a model transformation from t to s. This property can be useful in canceling the effects of a transformation [DBAS09].

Hermann et al. [HHK10] identify correctness, completeness and termination as the core requirements for a successful model transformation. A model transformation must ensure both syntactic and semantic correctness. In order to preserve and reuse the information in the source model, it needs to be complete as well. The completeness of a model transformation can be guaranteed by fully covering the source metamodel and the target metamodel [Vig09]. The termination requirement is provided by the transformation language interpreter, and so we will focus on correctness and completeness during the evaluation of the model transformations in Chapter 6.

2.3.8. Requirements for the application of MDD

In this section, we summarize the basic requirements that need to be satisfied for a successful application of MDD. These requirements are derived from the MDD literature and will be used to evaluate the MDD4MS framework in Chapter 3.

- **R-MDD.1.** Abstraction: Raising the level of abstraction of models to be closer to the problem domain and to be away from the implementation details [Sel06, MGS+11].
- **R-MDD.2.** Metamodeling: Defining the modeling languages formally with metamodels [AK03].
- **R-MDD.3.** Transformation: Facilitating user-defined mappings from models to other artifacts, including source code [AK03].
- **R-MDD.4.** Automation: Using computer technology to automatically generate models, modeling tools, source code, and documentation [Sel06].
- **R-MDD.5.** Generality: Describing the development process without relying on a specific technology, tool or platform [FRR09, MRB12].

2.4. Applying MDD in M&S

In the M&S field, the application of MDD principles is more clear than in software engineering, because a computer simulation model refers to the final executable source code in many cases directly or indirectly. Hence, code generation can be perceived as a model-to-model transformation as well. Besides, many simulation modeling languages exist that implement an abstract simulator.

Furthermore, system specification formalisms can be used to define formal and precise simulation models. For example, [ZPK00] present a generic framework for

M&S and various system specification formalisms based on general and mathematical systems theory. The entities of their framework are source system, experimental frame, model, and simulator. A model, a system, and an experimental frame are linked by the modeling relation, whereas a model and a simulator are linked by the simulation relation. An experimental frame is a specification of the conditions under which the system is observed or experimented with.

The Petri Nets formalism is also commonly used in the M&S field especially in the modeling of discrete event systems. Petri Nets are bipartite graphs and provide a mathematically rigorous modeling framework [Pet81]. They consist of places, transitions and directed arcs. Arcs run from a place to a transition or a transition to a place, never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places may contain a number of tokens. A transition of a Petri Net model is fired whenever there is a token at the start of all its input arcs. There are various types of Petri Nets, such as Timed Petri Nets, Stochastic Petri Nets and Colored Petri Nets.

As we stated in Section 2.2, the practice of M&S can be identified as a model based approach and an MDD method can be incorporated into the existing M&S methodologies. The MDD4MS framework provides such a method and it can be used with any underlying formalism. Applying MDD in M&S provides new capabilities for efficient development of reliable, error-free and maintainable simulation models. MDD supports formal validation and verification techniques and provides early detection of the flaws. Availability of the existing tools and techniques for both metamodeling and model transformations is one of the practical advantage of applying MDD.

MDD approach brings great advantages to M&S. It provides ways to formally define the models and modeling languages. Since models are defined in a good manner and free of implementation details, conceptual modelers and domain experts can understand the models more easily and they can play a direct role in simulation model development process. The simulation model implementation becomes more efficient and error-free since a big portion of the source code can be generated automatically. Most existing MDD research in M&S takes place with metamodeling and modeling environments. Although the research studies look very promising, especially conceptual model transformations have not been sufficiently studied yet in simulation field. Due to MDD has a broad range of contributors, there are various terms used in the M&S and software engineering communities. Before presenting the related work, various acronyms and approaches are explained in the next section.

2.4.1. Model based or model driven?

The 'model based (MB)' and 'model driven (MD)' terms and initials have been used in a variety of system and softwarerelated acronyms. Although there is a consensus that these approaches suggest the systematic use of models as the primary means of a process and facilitate the use of modeling languages, there is not a common understanding of the terminology. In this section, the definitions of the frequently used acronyms and the objectives of those approaches are presented.

Model based engineering (MBE) The main goal in MBE is to support an engineering process with various models during the design, integration, validation, verification, testing, documentation, and maintenance stages. Sometimes, a specific activity can be labeled as model based testing, model-based design (MBD), model-based integration, model based analysis, and so on. MBE originated in the 1980s in parallel with the evolution of the computer-aided design and MBD techniques. MBD provides a model-based visual method for designing complex systems, originally for designing embedded and distributed systems. MBE facilitates the use of domain specific modeling languages.

Model based systems engineering (MBSE) In systems engineering, the application of MBE principles is called model based systems engineering. MBSE provides the required insight in the analysis and design phases; it enhances better communications between the different participants of the project; and it enables effective management of the system complexity. A core idea of MBSE is to move the practice of systems engineering from a document-centric to a model-centric paradigm. INCOSE (International Council on Systems Engineering) has identified the institutionalized use of MBSE tools and techniques as an integral part of its vision.

Model driven engineering (MDE) MDE is a system development approach that uses models as the primary artifacts of the development process [Sch06b]. It introduces model transformations between different abstraction levels. In MDE, source models are transformed into destination models at different stages to (semi) automatically generate the final system. The main goal in MDE is increasing productivity through automated model transformations.

Model driven development (MDD) The application of the MDE principles in software engineering is called model driven development (MDD). MDD is also known as model driven software development (MDSD) or model driven software engineering (MDSE) [SVB+06]. The modern era of MDD started in the early 1990s and now offers a notable range of methods and tools.

2.4.2. Related work

MDD methods have been commonly used in the last decade in the simulation field [VdLM02, MZRM+08, RMdMZ09, ÇVS11, GPR13]. MDD approaches were introduced to M&S in 2001 when Bakshi et al. [BPL01] presented a practical

application of MIC into embedded system design and simulation. They provided a formal paradigm for specification of structural and behavioral aspects of embedded systems, an integrated model-based approach, and a unified software environment for embedded system design and simulation.

Vangheluwe et al. [VdLM02] introduced metamodeling and model transformation concepts into the theory of modeling and simulation in 2002. They present an approach to integrate three orthogonal directions of M&S research: multi-formalism modeling, model abstraction and metamodeling. De Lara and Vangheluwe [dLV02] present a tool for metamodeling and model transformations for simulation, namely AToM3. The usage of the tool is presented in [VdL02].

Tolk and Muguira [TM04] introduced the concepts of the MDA into the distributed simulation. They present the complementary ideas and methods to merge High Level Architecture (HLA) and Discrete Event System Specification (DEVS) within the MDA framework.

After the introduction of the initial ideas, there have been many efforts to use MDD concepts in M&S. In some cases, metamodeling is used to describe domain specific modeling languages only for one abstraction level [DdL09, LKPV03, LWQY09, TTH11, SM12]. In all of these studies, lower level code representation methods are defined via metamodeling and code generation facilities are provided by modeling environments. In other words, the higher level conceptual models are ignored and so MDD tools are only used for automatic code generation from a system specification. For example, Levytskyy et al. [LKPV03] present two DEVS metamodels that are used to automatically generate a tool that allows the graphical definition of DEVS models. The tool is capable of generating a representation suitable for the simulation by an external DEVS interpreter.

More unified solutions are presented by following either the MDA specification [GKMM06, DGRMP10, GPR13] or the MIC approach [TAO08, ÇVS10a, LDNA03]. Guiffard et al. [GKMM06] provide a study that aims at applying a model driven approach to the M&S in military domain. This work has been carried out in the context of a larger research program (High Performance Distributed Simulation and Models Reuse) sponsored by the DGA (Delegation Generale pour l'Armement) of the French Ministry of Defense. The paper presents a prototype implementation as well. The prototype demonstrates that the automated transformation from a source model to executable source code is possible. The authors state that the amount of work needed for writing correct and complete set of transformation rules is extremely large.

DAmbrogio et al. [DGRMP10] introduce a model driven approach for the development of DEVS simulation models. The approach enables the UML specification of DEVS models and automates the generation of DEVS simulations that make use of the DEVS/service oriented architecture (SOA) implementation. An example application for a basic queuing system is also presented.

Garro et al. [GPR13] propose an MDA-based process for agent-based modeling and simulation (MDA4ABMS) that uses the agent-modeling framework of the Eclipse agent modeling platform project. The Acore metamodel of the project is similar to EMF Ecore and defined in Ecore, but it provides higher level support for complex agents. MDA4ABMS process allows (automatically) producing platform specific simulation models starting from a platform-independent simulation model obtained on the basis of a CIM. Then, the source code can be automatically obtained with significant reduction of programming and implementation efforts.

Topcu et al. [TAO08] propose the Federation Architecture Metamodel (FAMM) for describing the architecture of a High Level Architecture (HLA) compliant federation. FAMM supports the behavioral description of federates based on live sequence charts and it is defined with metaGME. FAMM formalizes the standard HLA object model and federate interface specification.

Iba et al. [IMA04] propose a simulation model development process and present an example for agent based social simulations. The proposed process consists of three major phases: conceptual modeling, simulation design and verification. In the conceptual modeling phase, the modeler analyzes the target world and describes the conceptual model. In the simulation design phase, the modeler designs and implements the simulation model, which is executable on the provided simulation platform. The modeler translates the conceptual models into simulation models according to the suggested framework. In the verification phase, the modeler runs the simulation and inspects whether the simulation program is coded rightly. If necessary, the modeler returns to the first or second phase and modifies the models.

In addition to the MDA and MIC approaches, [RMdMZ09] present a UML-based DEVS simulation method which is placed in DEVS Unified Process (DUNIP). DUNIP proposes a process that uses the DEVS formalism as a basis for automated generation of models from various requirement specifications [MRMZ07, MZRM+08]. Mittal and Risco Martín [MRM13] present the DEVS Modeling Language (DEVSML) 2.0 stack which shows that an underlying DEVS metamodel provides model reusability, integration, and interoperability between different platforms. They also propose how the Department of Defense Architecture Framework (DoDAF) can be enhanced to incorporate executable models using the DUNIP process. This work is cited as executable architecture and the research about executable architectures in simulation field is also related to the MDD approach [GT10]. The term is generally used in the military domain and refers to an architecture that contains executable models.

All of the aforementioned efforts show the applicability of the MDD approach in the simulation field. However, to the best of our knowledge, there is no generic theoretical framework that provides guidance for formal model transformations while moving from a conceptual model to an executable simulation model. The MDD4MS framework provides a generic MDD framework for M&S which can

be incorporated into the existing M&S methodologies and presents a formalism independent solution via formalizing the steps. The next chapter presents the MDD4MS framework.

Chapter 3

MDD4MS: A Model Driven Development Framework for M&S

This chapter proposes a formal model driven development framework for modeling and simulation, which is called the MDD4MS framework. The MDD4MS framework presents an integrated approach to bridge the gaps between different steps of a simulation study by using metamodeling and model transformations. It mainly addresses the conceptual modeling and the simulation model development stages in M&S lifecycle and it can be incorporated into the existing methodologies for increasing the productivity, maintainability and quality of an M&S study. The M&S lifecycle presented in Chapter 1 is extended to clearly separate the formal models and computer simulation models.

3.1. The MDD4MS lifecycle

The MDD4MS framework proposes an extended lifecycle with a special focus on simulation model development stage. In this way, it can be embedded into the existing M&S methodologies easily. As shown in Figure 3.1 the model specification and model implementation stages are separated. A brief overview of the stages is already given in Section 1.3. In this section, the model-type outputs of the stages are highlighted and the simulation model development stage is refined. Since MDD4MS proposes a generic framework, the detailed activities in each stage are not explained. The MDD4MS framework defines a number of models on top of the final computer simulation model. It uses the concept of platform-independence and similar terms to PIM, PSM and PIM-to-PSM transformation from the MDA specification.

3.1.1. M&S study definition

If a problem owner identifies a need for a simulation study, he/she defines the purpose of the simulation study, his/her requirements, and problems or issues. This essentially includes setting the boundaries of the study and choosing the value system (key performance indicators) according to which the system will be assessed.

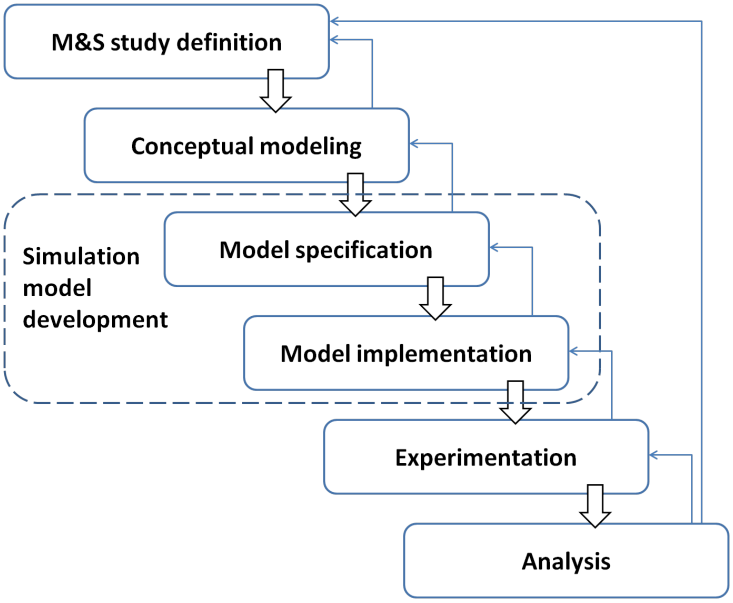


Figure 3.1: The MDD4MS lifecycle.

Requirements can be defined in a document or in a requirements model which can be specified with a requirements specification language [GMB94]. During the initial meetings with the problem owner and the conceptual modeler, a possible solution is suggested and outline of the simulation study is scheduled. The outcomes of this stage are presented in the M&S project plan.

3.1.2. Conceptual modeling

When the problem owner defines the requirements and starts the simulation study, the conceptual modeler makes a high-level abstraction of the real system or the future system according to a given worldview and prepares a conceptual model (CM). The CM is defined in a well-defined conceptual modeling language. A conceptual model serves as a bridge between the problem owner and the simulation modeler. The conceptual modeler prepares the CM by investigating the system and using his/her knowledge, available design patterns and the conceptual modeling language.

A simulation conceptual model (CM) refers to the non-executable higher level abstraction of the system under study. It represents the structure of the system and what will be modeled in the future executable simulation model.

3.1.3. Model specification

After the problem owner and the conceptual modeler agree on a CM, the simulation modeler transforms it into a formal model according to a system specification language for a certain formalism such as DEVS, Petri Nets, partial differential equations, or finite state automata. At this stage, the simulation modeler defines the system functionality without taking into account any specific platform on which the model will be later implemented. Hence, the formal model will be called a platform-independent simulation model (PISM). A PISM can be mapped to a PIM in MDA.

This is the stage of formalizing the CM so that mathematical analyses can be conducted and computational representation can be achieved. A PISM is a mathematical description of the processes and activities in the conceptual model, generally based on the available data. It is expected to be mathematically complete, in other words it can be simulated manually. However to be able to simulate it on a specific simulation platform it should be transformed to an executable model.

3.1.4. Model implementation

After the conceptual modeler and the simulation modeler agree on a PISM, the simulation programmer develops the platform-specific simulation model (PSSM). A PSSM is an implementation model of a PSIM for a specific platform. A PSSM can be mapped to a PSM in MDA. A final compilable and/or executable simulation model source code (SM) is automatically generated from a PSSM. PSSM and SM are at the same abstraction level with different views.

A PSSM should be specified in a modeling language which provides a higher level representation of a programming language. The SM is an executable source code generated/written in that programming language. The boundaries of the system in the surrounding environment are represented via parameters during the modeling process. At this stage, simulation model is validated and verified to test if it correctly and accurately represents the source system.

3.1.5. Experimentation

Once the executable SM is ready, the simulation expert designs the experiments and executes the SM on a simulator with the collected data. The run-time model includes the specific values for the model parameters and is called the experimental model (EM). Before the actual experiments, validation experimentations can be performed to validate if the input/output behavior of the simulation model matches to the purpose of the simulation study. Simulation results are generated during the experiments.

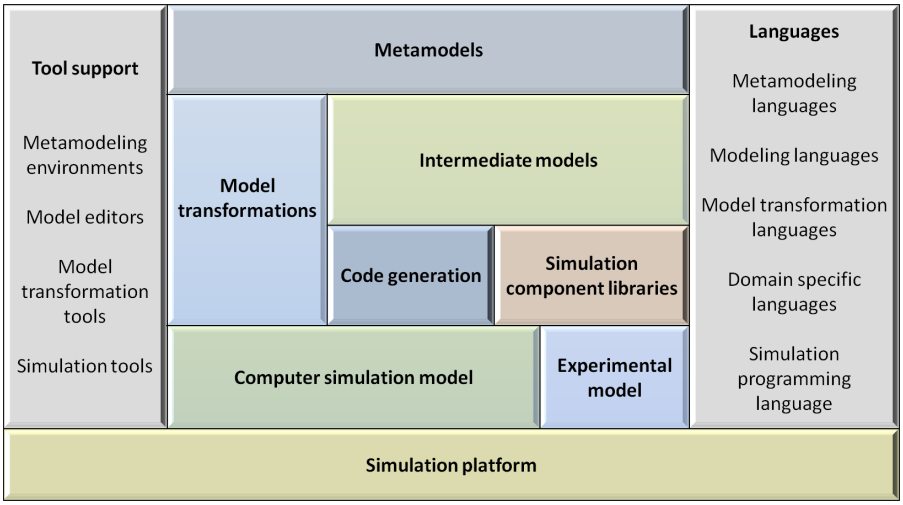


Figure 3.2: MDD4MS general architecture.

3.1.6. Analysis

The simulation analyst analyzes the experimentation results to extract maximum insight about the source system. The overall study and suggestions for future implementations are presented in the M&S report. Additional experiments can be performed if needed.

3.2. General architecture

The MDD4MS framework introduces model and metamodel definitions for various stages, model transformations between different models, methods to support the transformation steps, and a tool architecture for the overall process. MDD4MS introduces two ways to support the model transformations with domain-specific constructs. These are either using domain-specific languages or using domain-specific component libraries with the framework. Domain-specific and component based solutions are explained in Chapter 4 and 5 respectively, while the main framework is explained in this chapter. The general architecture of the MDD4MS framework is shown in Figure 3.2.

When the model-type outputs of the M&S lifecycle are analyzed, it is most likely that these models need to be specified in different modeling languages. The MDD4MS framework introduces metamodels for these languages in order to support model transformations. Hence, each model needs to be an instance of the corresponding metamodel. All of the metamodels are specified in a metamodeling language. Although it is practical to use the same metamodeling language, it is

also possible to use different metamodeling languages as far as there is a suitable model transformation language for each transformation.

The MDD4MS framework introduces model transformations between the different models of the M&S lifecycle in order to automatically generate some parts of the simulation model source code. Domain-specific languages or simulation component libraries can be used to increase the portion of the generated code. The simulation model is filled in with the specific experimentation parameters and executed on the simulation platform.

The MDD4MS framework focuses on simulation model development and fills in the gap between the conceptual modeling, model specification and model implementation stages. M&S study definition, experimentation and analysis stages are partially addressed. The requirements specification metamodel is out of the scope of this research and it is assumed that the requirements are defined in a document. Hence, the MDD4MS framework introduces three metamodels for CM, PISM and PSSM.

- CMmetamodel represents the grammar of a conceptual modeling language,
- PISMmetamodel represents the grammar of a system specification formalism,
- PSSMmetamodel represents the grammar of a simulation model programming language.

Where,

- CM is an instance of the CMmetamodel,
- PISM is an instance of the PISMmetamodel,
- PSSM is an instance of the PSSMmetamodel.

The metamodels are expected to be specified in a metamodeling environment, where model editors can be generated automatically. By using the model editors, the models are specified as the instances of the metamodels. After introducing the metamodels, the following metamodel based model transformations are proposed:

- CMtoPISM transformation is a M2M transformation from CM to PISM.
- PISMtoPSSM transformation is a M2M transformation from PISM to PSSM.
- PSSMtoSM transformation is a M2M transformation from PSSM to SM.

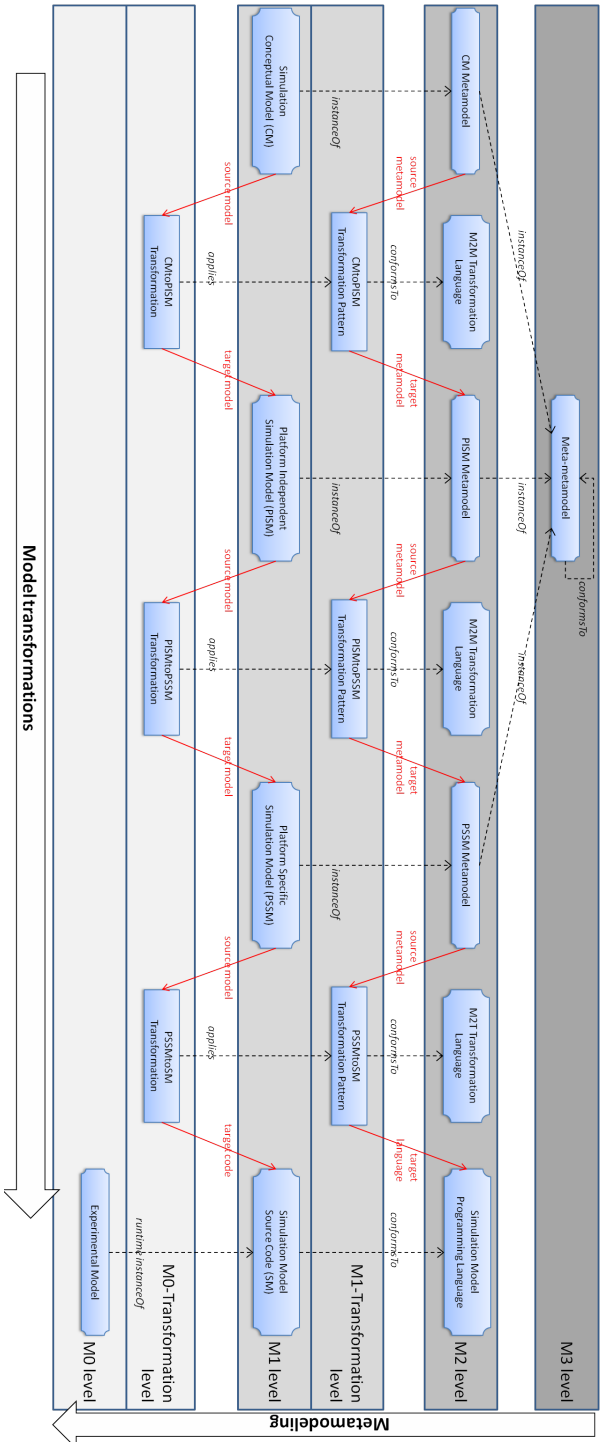


Figure 3.3: MDD4MS framework: models, metamodels and model transformations [ÇMVS13].

Figure 3.3 illustrates the proposed framework with one meta-metamodel. There are three metamodeling stacks, which are presented vertically in the figure. Model transformations lay on an orthogonal axis to metamodeling as shown in the figure.

In CMtoPISM and PISMtoPSSM transformations, extra knowledge needs to be added to obtain a full model. Otherwise the target model can only be partially generated. While moving from a CM to PISM, the detailed behavior of the system should be explained; and while moving from a PISM to PSSM, the implementation details should be explained. CMtoPISM transformation contains domain-specific constructs in order to add execution semantics. PSSMtoSM transformation is generally a one to one transformation for a programming language and its metamodel.

Model transformations enable the reuse of information that was once described in a model. Defining precise metamodels and writing good model transformation rules are the challenging activities of MDD. Various model transformations can be defined for the same source model, so it is important to define a good model transformation or choose the best existing one for a successful MDD process. To increase the effectiveness of model transformations, domain-specific modeling languages and component libraries are used in MDD applications. In this case, the transformations can be written in a more generic and compact way since the domain knowledge is already added via the language elements or components.

A practical application of the MDD4MS framework is an MDD process and it is called an MDD4MS process. Although we propose an MDD4MS process with three intermediate models and one final executable model, it is possible to increase the number of models in the framework. MDD4MS clearly separates the conceptual modeling, model formulation (specification), implementation and model coding with the associated models CM, PISM, PSSM, and SM. However, according to the different needs in simulation projects, it is also possible to merge some stages by using the same modeling language to specify the different models, e.g., same language can be used for CM and PISM stages in small scale projects. Then, the higher level model becomes less detailed. The lower level model is expected to include the missing information in the higher level model. For example, an incomplete visual DEVS diagram can serve as a conceptual model, if the problem owner is familiar with DEVS. Besides, different model transformation patterns can be defined for a metamodel, e.g., a CM can be used to generate multiple PISMs, and a PISM can be used to generate multiple PSSMs. Figure 3.4 shows a sample workflow for the simulation model development stage of an MDD4MS process.

3.3. Theory of MDD

This section explains the theoretical underpinnings of modeling, metamodeling and model transformations. The theory is based on the information given in Section 2.3. Although, there have been related work in the software engineering literature for mathematical explanation of MDD concepts [Fav04, Küh06, JB06, JS09], we could not find a sound and comprehensive reference that covers all of the definitions.

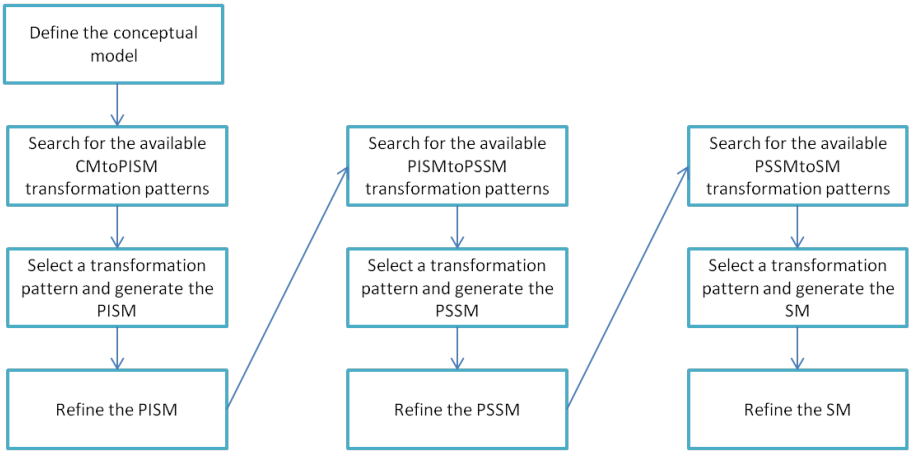


Figure 3.4: A sample workflow for the simulation model development stage.

Yet, the preliminary definitions and ideas helped us to formalize the steps in MDD processes.

3.3.1. Modeling

Modeling is the process of representing a source system for a specific purpose in a form that is ultimately useful for an interpreter. The concrete form that represents the system is called the model. A model is developed for a purpose related to the source system [Kli69, Sha75, Ack78]. This purpose can be achieved by executing or interpreting the model and gaining knowledge that relates to the source system. The interpretation can only be validated in a given context. The context includes the purpose of the modeling process, information about the surrounding environment of the source system, and the constraints, assumptions and facts that affect the modeling process. A context can be formally defined and specified in a model as well [TACS02]. The following primitive terms and relations are defined:

- S is the infinite set of all source systems.
  Variables such as "$s, s_1, s_2, ...$" range over S.
- C is the infinite set of all contexts.
  Variables such as "$c, c_1, c_2, ...$" range over C.
- L is the infinite set of all formal languages.
  Variables such as "$l, l_1, l_2, ...$" range over L.

The usual set operations are applicable to S, C and L such as union, intersection, difference, complement, subset, proper subset, Cartesian product and powerset.

The + function $c_1 + c_2 : C \times C \rightarrow C$ is used to represent the composition of two contexts, where $c_1 + c_1 = c_1$, $c_1 + c_2 = c_2 + c_1$ and $c_1 \leq c_1 + c_2$.

The grammar notion of the formal language theory (see Appendix B) is used to define the abstract syntax of a language and it is accepted that a default concrete syntax is provided by the metalanguage. The expression generated with the concrete syntax is called the model and it is often in a structured textual form. Then, the following definitions and axioms are proposed.

Definition 1 (Model). *Let $g = \{T, N, I, P\}$ be a grammar, and the language that g generates is $l(g)$. If an expression $m \in l(g)$ is a representation of a source system s within a context c, then m is said to be a model of s in c. The infinite set of all models is denoted with M. Variables such as "$m, m_1, m_2, ...$" range over M. The ternary relation 'model-of' is denoted with μ: $M \times S \times C$.*

Definition 2 (Conforms-to relation). *If an expression $m \in l(g)$ and $m \in M$, then the language l is a modeling language and m conforms-to l. The binary relation 'conforms-to' is denoted with γ: $M \times L$.*

Axiom 1 (transitive-μ). $\mu(x, y, c_1) \wedge \mu(y, z, c_2) \Rightarrow \mu(x, z, c_1 + c_2)$.

Figure 3.5 shows the relationships between a model, a source and a language. Figure 3.6 illustrates the Axiom 1.

Although the terminal symbols of the grammar provide a default concrete syntax (primary view), it is possible to define other concrete syntax mappings (secondary views) for some reasons. For example, it is very common that a model has a textual view and a diagrammatic view in computer science. We suggest using an extended grammar to define a mapping from the default concrete syntax to a set of concrete symbols. In this way, all of the possible final or intermediate productions of a given grammar with both terminal and non-terminal symbols can be included in the mapping. The language defined with the extended grammar allows defining composed modeling elements for a modeling language. The most important thing is that the primary view has all the specified information and the secondary views will be syntactically either equivalent to or weaker than it.

Definition 3 (Extended grammar). *Let $g = \{T, N, I, P\}$ be a grammar, and $l(g)$ be a modeling language. Let $\widehat{g} = \{T \cup N, N, I, P\}$ be another grammar, and $\widehat{l}(\widehat{g})$ be all expressions generated by $\widehat{g}$. $\widehat{l}$ includes all of the possible expressions with both terminal and non-terminal symbols of g. Hence, $l \subseteq \widehat{l}$. The grammar $\widehat{g}$ is the extended grammar of g.*

Definition 4 (Concrete syntax mapping). *Let $g = \{T, N, I, P\}$ be a grammar, and $l(g)$ be a modeling language. A concrete syntax mapping for $l(g)$ is a binary relation from $\widehat{l}(\widehat{g})$ to cs, where cs is a set of concrete symbols, and the relation is denoted as $\psi(\widehat{l}(\widehat{g}), cs)$.*

Definition 5 (Secondary view). *Let $m \in M$ be a model that conforms-to a modeling language $l(g)$. For a given set of concrete symbols cs, if m can be mapped to an*

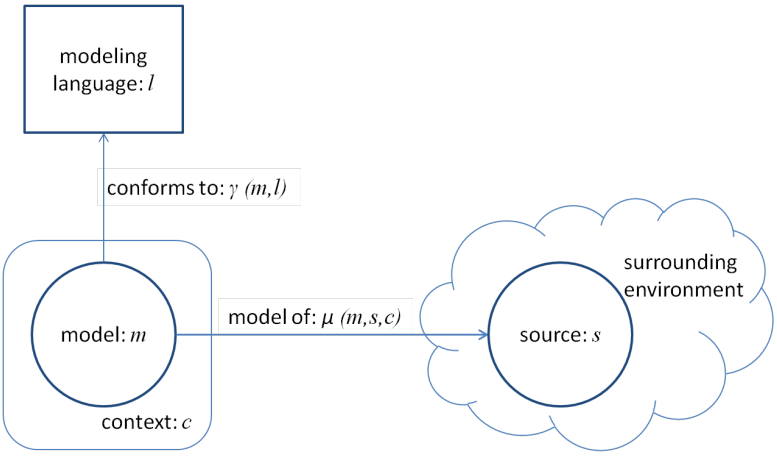


Figure 3.5: The relationships between model, source and language.

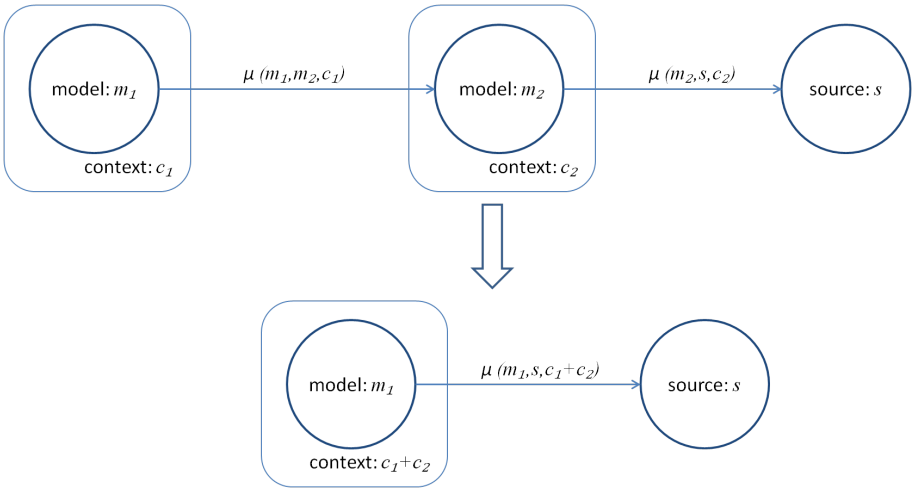


Figure 3.6: Illustration of the Axiom 1.

expression $v \in cs^*$, by using a concrete syntax mapping $\psi(\widehat{l(g)}, cs)$, then v is a secondary view of the model m. cs^* is the infinite set of all expressions, which can be obtained by composing zero or more symbols from cs. The infinite set of all secondary views is denoted with V. Variables such as "$v, v_1, v_2, ...$" range over V.

In some cases, a model editor is capable of hiding some parts of a model in a particular aspect such as hiding a specific type of connection in a model. In this case, the concrete syntax remains unchanged. On the other hand, if a concrete syntax mapping is defined then the editor shows a different view of the model such as changing the shape of a specific type of element in a model. Although any subsequent view of a model is based on the default concrete syntax, semantically it can be more powerful than the model itself in a domain.

3.3.2. Metamodeling

Metamodeling is the process of specifying a grammar of a modeling language in the form of a model, which in turn can be used to specify models in that language. Hence, metamodeling is a modeling process where the source is a grammar. The following definitions can be derived from the earlier ones.

Definition 6 (Metamodel)**.** *Let $mm \in M$ be a model and $\exists\mu(mm, s, c)$, where $s \in S$ and $c \in C$. A model mm is a metamodel if, and only if, s is a grammar. The infinite set of all metamodels is denoted with M', where $M' \subset M$. Variables such as "$mm, mm_1, mm_2, ...$" range over M'.*

Definition 7 (Metamodeling language)**.** *For any $mm \in M'$ and $\gamma(mm, l)$, the language l is a metamodeling language. The infinite set of all metamodeling languages is denoted with L', where $L' \subset L$. Variables such as "$l', l'_1, l'_2, ...$" range over L'.*

If a model m is an instance of a metamodel mm, where mm is a model of a grammar g, then the model m *conforms-to* $l(g)$. Figure 3.7 illustrates the relations between a model and a metamodel.

Definition 8 (Instance-of relation)**.** *Let $mm \in M'$ be a metamodel, $m \in M$ be a model, and $\gamma(m, l(mm))$. m is an instance-of mm if, and only if, every element of m is an instance of some element in mm. The binary function 'instance-of' is denoted with $\tau(m) : M \to M'$.*

Axiom 2 (Formalized conforms-to)**.** $(\tau(x) = y) \Rightarrow \gamma(x, l(y))$ *(by definition 8).*

3.3.3. Model transformations

A formal transformation requires that the models are specified in well-defined modeling languages and the rules are defined with a model transformation language. A model transformation is performed with a model transformation pattern. According to the requirements given in Section 2.3.7, we assume that a model transformation is correct and complete so that it preserves the modeling relation.

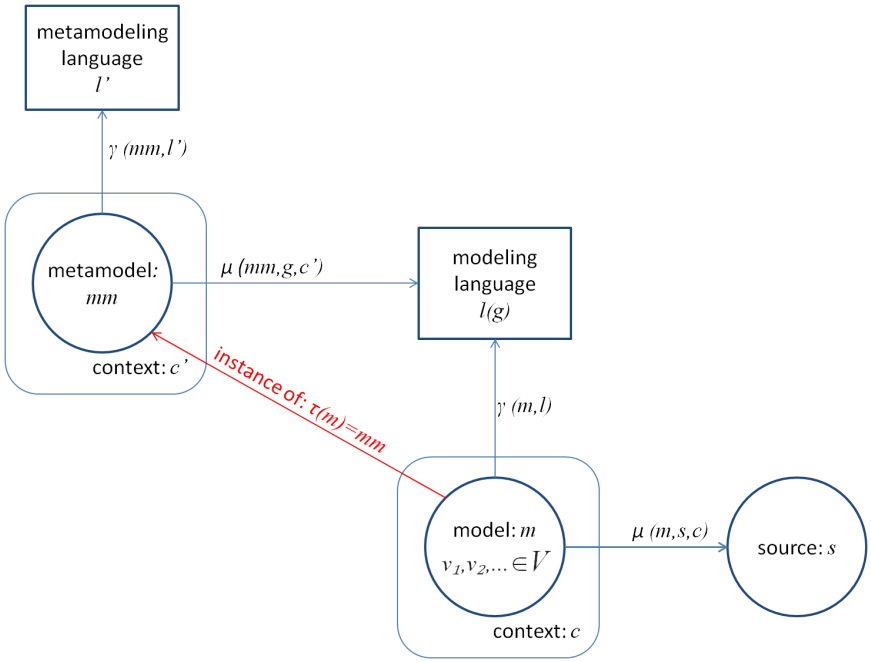


Figure 3.7: Metamodeling.

Definition 9 (Model transformation pattern). *A model transformation pattern p is defined as a triple $p = \{l_x, l_y, r\}$, where*

l_x is the source modeling language,

l_y is the target language (any language such as a programming language, a modeling language or any well-defined language),

r is a finite set of transformation rules from l_x to l_y which are defined with a model transformation language.

The infinite set of all model transformation patterns is denoted with P. Variables such as "$p, p_1, p_2, ...$" range over P.

Definition 10 (Formal model transformation). *Let $m \in M$ be a model that conforms to l_1, and $p = \{l_1, l_2, r\}$ be a model transformation pattern.*

If an expression $e \in l_2$ is derived from m by applying p, then the derivation is called a formal model transformation. It is said that m is transformed-to e by applying p and denote it as $\theta(m, p) = e$.

Definition 11 (Formal model-to-model transformation). *If $\theta(x, p) = y$ and $y \in M$, then the transformation is called a formal model-to-model transformation.*

Axiom 3. *If the model transformation pattern p preserves the system related information in the source model, then*

$(\theta(x, p) = y) \wedge \mu(x, s, c) \wedge (y \in M) \Rightarrow \mu(y, s, c + p)$.

Figure 3.8 shows the relationships in a model-to-model transformation. Figure 3.9 illustrates the Axiom 3.

Definition 12 (Code generation). *If $\theta(x, p) = y$ and y is a source code in a software programming language, then the transformation is called code generation.*

Axiom 4. $(\theta(x, p) = y) \wedge (p = \{l_1, l_2, r\}) \Rightarrow \gamma(x, l_1)$ *(by definition 10).*

Axiom 5. $(\theta(x, p) = y) \wedge (y \in M) \wedge (p = \{l_1, l_2, r\}) \Rightarrow \gamma(y, l_2)$ *(by definition 10 and definition 11).*

The following axioms are defined for the metamodel based model transformations.

Axiom 6. $(\theta(x, p) = y) \wedge (p = \{l(mm_1), l_2, r\}) \wedge (mm_1 \in M') \Rightarrow (\tau(x) = mm_1)$

(by definition 10).

Axiom 7. $(\theta(x, p) = y) \wedge (y \in M) \wedge (p = \{l_1, l(mm_2), r\}) \wedge (mm_2 \in M') \Rightarrow$

$(\tau(y) = mm_2)$ *(by definition 10 and definition 11).*

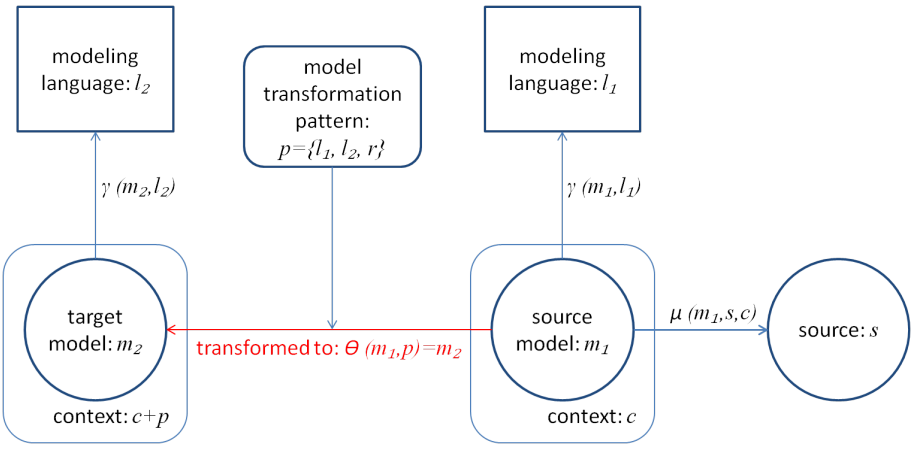


Figure 3.8: The relationships in a model-to-model transformation.

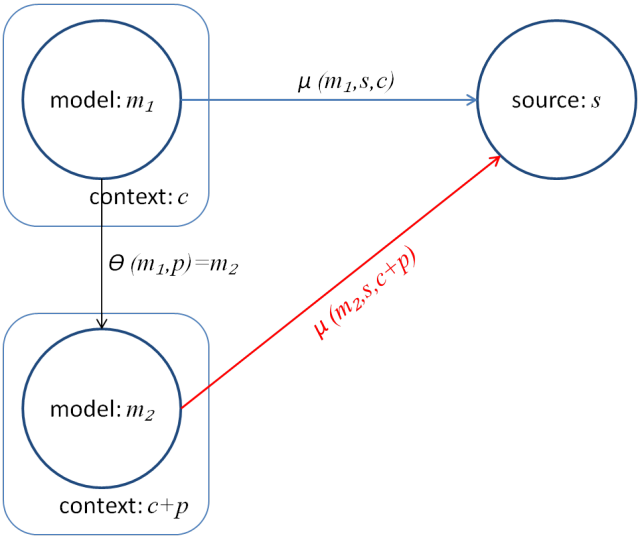


Figure 3.9: Illustration of the Axiom 3.

3.3.4. MDD process

By using the definitions given in the previous sections, we define an MDD process as:

Definition 13 (MDD process)**.** *A model driven development process for a software application development is defined as a tuple*

$$mdd = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, sc, TO\}$$

where

n *is the number of the intermediate models,*

$MML = \{l'_0, l'_1, ..., l'_{n-1}\}$ *is an ordered set of metamodeling languages (can be defined with meta-metamodels),*

$ML = \{l_0(mm_0), l_1(mm_1), ..., l_{n-1}(mm_{n-1}) \mid \gamma(mm_i, l'_i)(0 \leq i < n)\}$ *is an ordered set of modeling languages (preferably defined with metamodels),*

$MO = \{m_0, m_1, ..., m_{n-1} \mid \gamma(m_i, l_i)(0 \leq i < n)\}$ *is an ordered set of models,* m_0 *is the initial model and* m_{n-1} *is the final model,*

SL *is a set of supplementary languages (including at least a model transformation language for writing transformation rules),*

pl *is a programming language for final code generation,*

$MTP = \{p_0, p_1, ..., p_{n-2}, p_{n-1}\}$ *is a set of formal model transformation patterns, where* p_i *is a model-to-model transformation pattern,* p_{n-1} *is a code generation pattern, and* $(p_{n-1} = \{l_{n-1}(mm_{n-1}), pl, r\}) \in MTP$ *(including at least the final code generation pattern),*

STP *is a set of other supplementary model transformation patterns,*

$MT = \{(\theta(x, p) = y) \mid (x \in M) \wedge (p \in MTP)\}$ *is a set of formal model transformations, where* $(\theta(m_{n-1}, p_{n-1}) = sc) \in MT$ *(including at least the final code generation),*

sc *is the final source code,*

TO *is a set of tools to ease the activities.*

An MDD process is supposed to have an initial model, a number of intermediate models and final source code. The aim is to obtain a large part of the intermediate models and the final code through successive model transformations. An MDD process supports model continuity by formal model transformations.

Definition 14 (Model continuity)**.** *Let* mdd *be an MDD process,* m_0 *be the initial model of this process and* $\mu(m_0, s, c)$*. It is said that model continuity is obtained in*

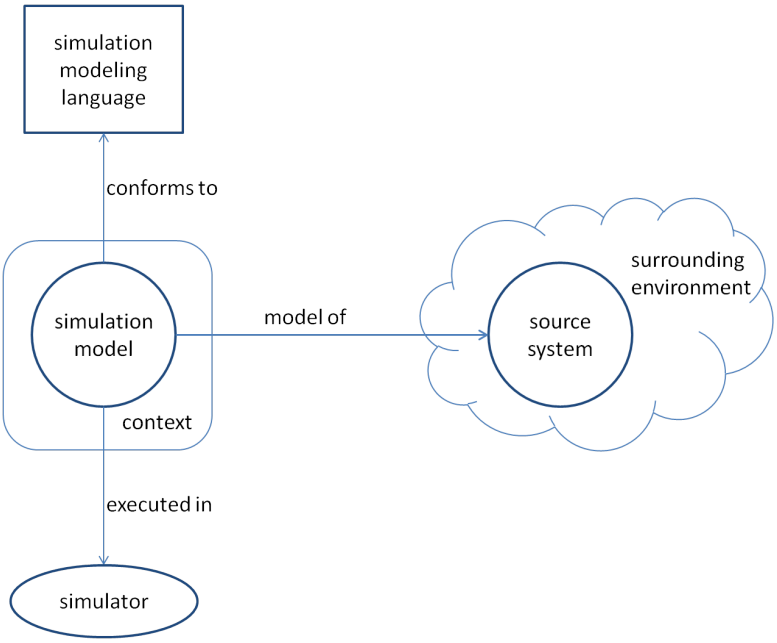


Figure 3.10: Modeling and simulation in general.

an mdd process if, and only if $n \geq 2$ and $\mu(m_{final}, s, c{+}x)$, where m_{final} is the final model of the MDD process, it is generated through formal model transformations, and it preserves the modeling relation.

3.4. Theory of the MDD4MS framework

Following the earlier definitions, modeling for computer simulation is the process of representing a system for a specific purpose in a form that is executable within a simulator. Figure 3.10 shows the basic concepts and relations based on the general modeling principles.

The main difference from a general modeling process is that the model is interpreted (i.e. executed) within a computerized model interpreter (i.e. simulator). Hence, a simulation model is executable in a context within a simulator. A simulator is a computer program which may be executed on a computer platform or may be embedded directly into a hardware platform. The output of the simulator is called the simulation results. Due to the fact that the main objective of a simulation model is being simulated, a computer simulation model needs to be specified in a programming language that provides or can be extended to provide simulation capabilities (preferably with a suitable model editor). Simulation is the process of conducting experiments with a model so that the behavior of the system is

simulated over time. We propose the following definitions based on the MDD concepts.

Definition 15 (Simulation model)**.** *Let $m \in M$ be a model, $s \in S$ be a source system, and $\mu(m, s, c)$. m is a simulation model if, and only if, there exists a simulator that can simulate m over time.*

Definition 16 (Simulation modeling language)**.** *Let $l(g)$ be a modeling language. The language $l(g)$ is a simulation modeling language if, and only if, there exists a simulation model m such that $\gamma(m, l(g))$.*

Due to the fact that a computer simulation model refers to the final executable source code in many cases directly or indirectly, code generation is accepted to be a model-to-model transformation in the MDD4MS framework. An implementation pattern for applying MDD concepts in M&S domain is given as follows:

Definition 17 (MDD4MS process with three intermediate models)**.** *An MDD4MS process with three intermediate models is a specialized MDD process as*

$$mdd4ms = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}$$

where

$n = 3$ $(CM, PISM, PSSM)$,

$MML = \{l'_0, l'_1, l'_2\}$ *is an ordered set of metamodeling languages (can be defined with meta-metamodels),*

$ML = \{l_0(CMmetamodel), l_1(PISMmetamodel), l_2(PSSMmetamodel)\}$ *such that*

- $\gamma(CMmetamodel, l'_0)$,
- $\gamma(PISMmetamodel, l'_1)$,
- $\gamma(PSSMmetamodel, l'_2)$,

$MO = \{CM, PISM, PSSM\}$ *such that CM is the initial model, PSSM is the final model, and*

- $\tau(CM) = CMmetamodel$,
- $\tau(PISM) = PISMmetamodel$,
- $\tau(PSSM) = PSSMmetamodel$,

SL *is a set of model transformation languages,*

pl *is a programming language with simulation capabilities,*

$MTP = \{p_{cm}, p_{pism}, p_{pssm}\}$ *such that*

- p_{cm} = $\{l_0(CMmetamodel), l_1(PISMmetamodel), r_0\}$,
- p_{pism} = $\{l_1(PISMmetamodel), l_2(PSSMmetamodel), r_1\}$,
- p_{pssm} = $\{l_2(PSSMmetamodel), pl, r_2\}$,

STP *is a set of other supplementary formal model transformation patterns,*

MT = $\{(\theta(CM, p_{cm})$ = $PISM)$, $(\theta(PISM, p_{pism})$ = $PSSM)$,

$(\theta(PSSM, p_{pssm})$ = $SM)\}$,

SM *is the final executable simulation model,*

TO *a set of tools to ease the activities.*

Theorem 1. *An MDD4MS process with three intermediate models (performed according to the definition 17) obtains model continuity.*

Proof. For a given

$$mdd4ms = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}$$

where $n = 3$ and $MO = \{CM, PISM, PSSM\}$, according to definition 17, we have:

- $CM, PISM, PSSM \in M$,
- p_{cm} = $\{l_0(CMmetamodel), l_1(PISMmetamodel), r_0\}$,
- p_{pism} = $\{l_1(PISMmetamodel), l_2(PSSMmetamodel), r_1\}$,
- p_{pssm} = $\{l_2(PSSMmetamodel), pl, r_2\}$
- $\theta(CM, p_{cm}) = PISM$,
- $\theta(PISM, p_{pism}) = PSSM$,
- $\theta(PSSM, p_{pssm}) = SM$,
- SM is the final executable simulation model.

We assume that CM is the initial model, $\mu(CM, s, c)$ and s is a system. As well as, model transformations are correct and complete. Although, $PSSM$ is the final model in software engineering, we accept that SM is the final model in M&S. Figure 3.11 illustrates the Theorem 1.

1. $(\theta(CM, p_{cm}) = PISM) \wedge \mu(CM, s, c) \wedge PISM \in M \Rightarrow \mu(PISM, s, c + p_{cm})$ (by axiom 3).

2. $(\theta(PISM, p_{pism}) = PSSM) \wedge \mu(PISM, s, c + p_{cm}) \wedge PSSM \in M \Rightarrow$ $\mu(PSSM, s, c + p_{cm} + p_{pism})$ (by 1 and axiom 3).

3. If SM is a simulation model, then $SM \in M$ (by definition 15).

4. $(\theta(PSSM, p_{pssm}) = SM) \wedge SM \in M \wedge \mu(PSSM, s, c + p_{cm} + p_{pism}) \Rightarrow$ $\mu(SM, s, c + p_{cm} + p_{pism} + p_{pssm})$ (by 1, 2, 3 and axiom 3).

5. $n \geq 2 \wedge \mu(SM, s, c + p_{cm} + p_{pism} + p_{pssm}) \Rightarrow$ *mdd4ms process obtains model continuity* (by definition 14).

□

3.5. Tool architecture for MDD4MS

The most notable advantages of MDD are rapid software development and increased productivity. Hence, computerized tool support is very important in MDD approaches. In order to support the M&S lifecycle, a tool architecture for the MDD4MS lifecycle is proposed in Figure 3.12. A full-featured metamodeling environment, a set of model editors and a set of model transformation tools are required.

The MDD4MS framework introduces new roles into the M&S field as metamodeler and transformation rule writer. The following roles are defined whereas a participant can have more than one role: Problem owner, requirements analyst, conceptual modeler, simulation modeler, simulation programmer, metamodeler, modeling tool developer, transformation rule writer, and simulation expert/analyst.

As explained before, metamodeling environments can generate the modeling tools automatically. Many tools have decorator facilities for nicer visualization of the models and some model verification facilities. Graphical modeling tools for simulation conceptual modeling, simulation model specification and simulation model implementation can be generated automatically by using the available MDD tools. The auto-generated modeling tools can provide a drag and drop model editor with a drawing palette that shows the modeling elements in the metamodel; some basic file operations (new, open, save, close); some editing functions (editing the properties, cut, copy, paste, delete, move, zoom in, zoom out, undo, redo); switching between multiple windows (symbolic language elements, properties, model explorer, etc.); toolbars and menus. Sometimes, the metamodeling environment can support the generated modeling tool with extra facilities such as writing a model transformer-/interpreter for the models developed with that tool. Model transformation tools perform the M2M or M2T transformations.

A simulator can be seen as an interpreter for a programming language that executes the SM. If necessary, it compiles the source code first. In this case, the

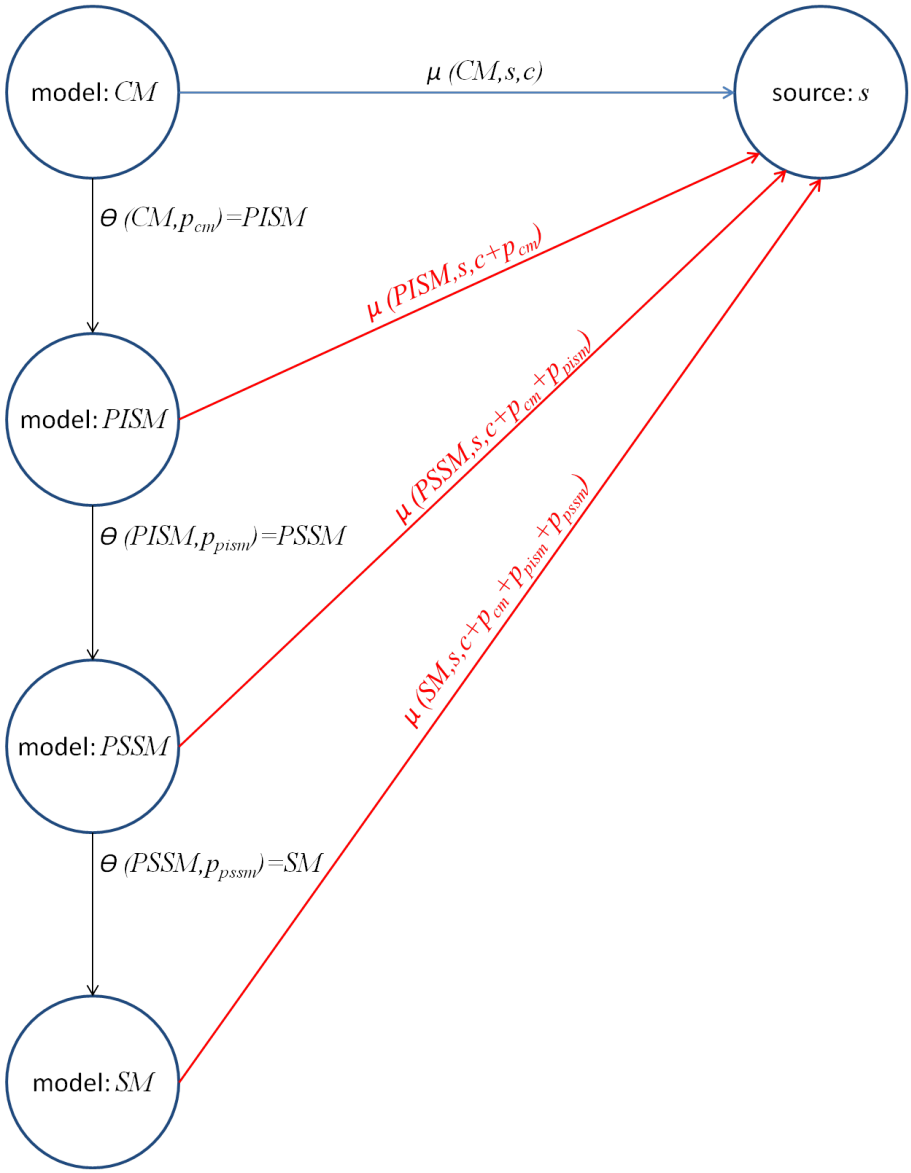


Figure 3.11: Illustration of the Theorem 1.

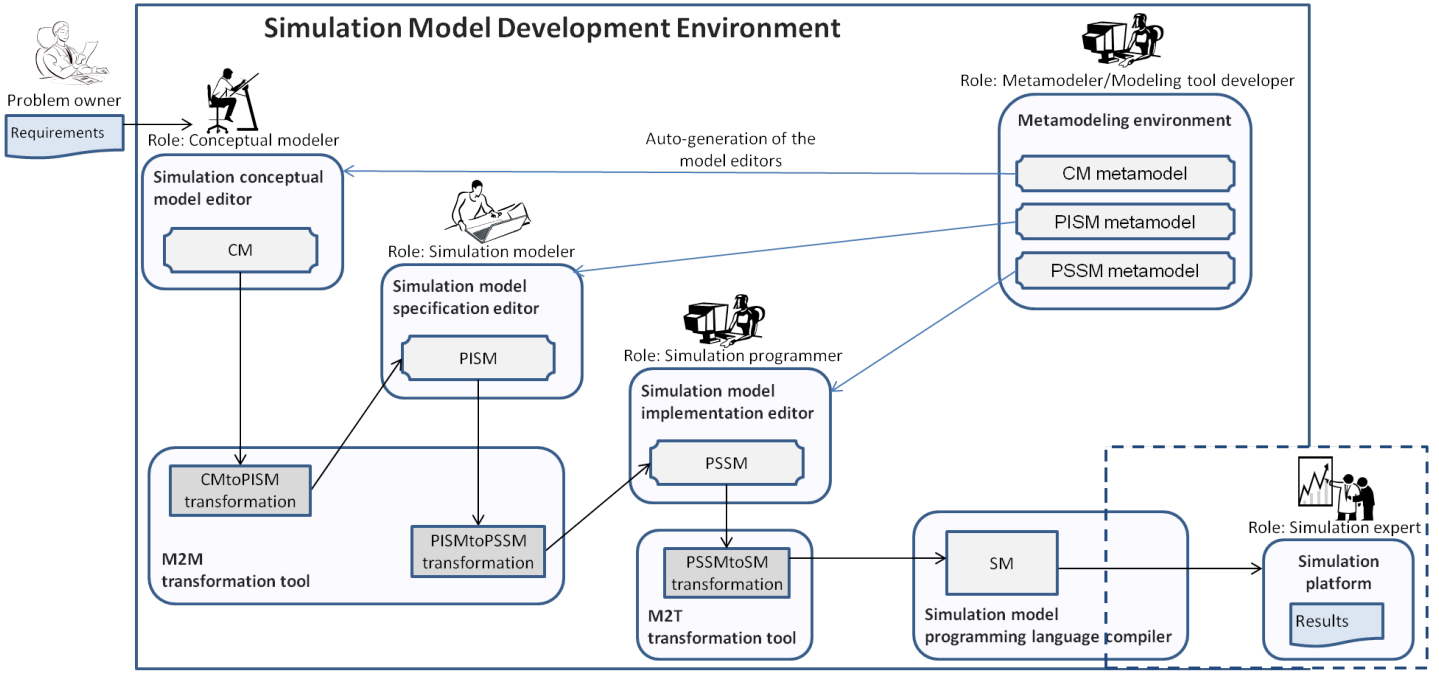


Figure 3.12: Tool architecture for the MDD4MS framework.

simulator includes a compiler as well. The simulator can be either included or excluded from the modeling tools. If it is a general purpose programming language interpreter, then it is better to keep it apart from the modeling tools. An IDE that combines the aforementioned graphical modeling tools and model transformers is called a simulation model development environment. The intended users of the environment are conceptual modelers, simulation modelers, simulation programmers, simulation experts or general users with basic M&S knowledge. Finally, Table 3.1 provides a checklist for applying MDD4MS in practice. The required activities are listed in the table and the name of the artifacts or chosen methods/languages can be written.

3.6. Evaluation of the proposed MDD application

The proposed MDD4MS framework is a conceptual application of the MDD principles in the M&S field. Hence, we evaluate the framework according to the requirements given in Section 2.3.8.

- **R-MDD.1.** Abstraction: The MDD4MS framework suggests the use of at least four models which are CM, PISM, PSSM and SM. The models are at different abstraction levels, for example CM is more closer to the problem domain without execution semantics while PISM includes the execution semantics but not the implementation details, and PSSM includes both. Hence, the MDD4MS framework satisfies the abstraction requirement.

- **R-MDD.2.** Metamodeling: The framework uses metamodeling for language definitions and suggests the use of at least three metamodels which are CM, PISM and PSSM metamodels. So, the MDD4MS framework satisfies the metamodeling requirement.

- **R-MDD.3.** Transformation: The framework defines three model transformations from CM to PISM, PISM to PSSM, and PSSM to SM. A transformation needs to be defined by transformation rules with a transformation language. So, the MDD4MS framework satisfies the transformation requirement.

- **R-MDD.4.** Automation: The framework presents a tool architecture and suggests using existing metamodeling environments to automatically generate models and source code, as well as using model transformation tools to generate models. Hence, the MDD4MS framework satisfies the automation requirement.

- **R-MDD.5.** Generality: The framework proposes a generic simulation model development method which can be incorporated into the existing methodologies. The framework is both formalism independent and platform independent, but it can be tailored according to a specific platform or technology. So, the MDD4MS framework satisfies the generality requirement.

Table 3.1: Checklist for applying the MDD4MS framework.

Activities	**Done? Y/N**	**artifact/ chosen method**	**Notes**
Choose the conceptual modeling language			
Choose the system specification formalism			
Choose the simulation programming language			
Choose a metamodeling language			
Choose a metamodeling environment			
Define/choose the simulation conceptual modeling metamodel (CMmetamodel)			
Define/choose the simulation model specification metamodel (PISMmetamodel)			
Define/choose the simulation model implementation metamodel (PSSMmetamodel)			
Choose a M2M transformation language			
Choose a M2T transformation language			
Choose a M2M transformation tool			
Choose a M2T transformation tool			
Define/choose the CM-to-PISM transformation			
Define/choose the PISM-to-PSSM transformation			
Define/choose the PSSM-to-Code transformation			
Generate/choose the simulation conceptual model editor			
Generate/choose the simulation model specification editor			
Generate/choose the simulation model implementation editor			
Choose a simulation platform			
Specify the CM			
Generate and refine the PISM			
Generate and refine the PSSM			
Generate and refine the SM			
Design experiments			
Execute the SM			

As a result, our MDD adoption satisfies all the requirements and so MDD4MS framework successfully applies the MDD concepts.

Chapter 4

Using Domain Specific Languages with the MDD4MS Framework

This chapter explains how domain specific languages can be added into the MDD4MS framework and how they can be utilized to support model transformations.

4.1. Domain specific languages

A domain specific language (DSL) is a language designed for a particular domain which can be either graphical or textual. The elements of a DSL represent the concepts of a particular domain and they enable the modeler to focus on the detailed aspects of the system. The opposite is a general purpose language which provides more general concepts in a discipline. For example, BPMN is a DSL for business process domain while UML is a general purpose modeling language for software engineering.

It is important to support the modeling process in a domain by graphical modeling tools which will help the modelers to construct their models faster, better and in a more reliable way. Hence, MDD tools are commonly used to develop modeling tools for DSLs. In MDD, a DSL is specified with a metamodel; and so the metamodeling process is sometimes called as Domain Specific Modeling (DSM). Once a DSL is specified with a metamodel then the instances of this metamodel are called domain specific models. In this case, a DSL is also known as a domain specific modeling language (DSML).

Please note that, the term 'domain-specific' is generally used for expressing different application domains such as logistics, health care, airports, container terminals, etc. in the M&S field. However, in the MDD context, a DSL is not necessarily designed for such kind of an application domain. For example, SQL (Structured Query Language) and HTML (Hyper Text Markup Language) are commonly referred as DSLs [vDKV00]. Hence, while a simulation programming language can be considered as a DSL from the point of an MDD expert, it is not domain-specific from the point of a simulation expert since it can be used for any kind of application domain.

4.2. Adding domain specific languages into the MDD4MS

Adding domain specific constructs into the MDD4MS framework has significant effect on model transformations. Domain specific constructs are more expressive in a particular domain and so model transformations can produce more precise and detailed models. It is also a way for defining specific transformation rules for a domain. DSLs can be used at any stage in the MDD4MS framework by using metamodeling.

When there is a metamodel for a stage in an MDD4MS application it is also possible to add new domain specific constructs to the related stage. In this case, to be clear in our terminology, we will use 'Domain-Specific metamodel (DS-metamodel)' term for a metamodel in a specific application domain such as logistics. For example, if we have a DEVS metamodel and we would like to have more domain specific DEVS constructs for logistics, we develop DEVS-logistics domain specific metamodel. A domain specific metamodel for an application domain in simulation can be added into the MDD4MS in three ways (when there is already a metamodel for the related stage).

4.2.1. Adding a new metamodeling layer

The first one is adding a new metamodeling layer between M1 and M2. Although this seems possible in theory, it is not very easy in practice. Because in order to achieve this, M2-level metamodels need to have metamodeling capabilities, which means that extra work should be done. For example everything in M3-level meta-metamodel should be duplicated in each M2-level metamodel. In this case M2-level metamodel becomes a meta-metamodel, so that a new metamodel can be generated from that. Necessarily, the metamodeling levels need to be renumbered.

OMG introduces a very useful concept of a UML profile to specify a domain specific model which can be instantiated for specifying models in that domain. The Profiles package included in UML 2.0 provides an extension mechanism through stereotypes, constraints and tagged values. A different notation and semantics can be defined for already existing elements. Although this is not exactly adding a new layer, it is the only available practical implementation.

4.2.2. Defining a new metamodel for M2 layer

The second way is making a new metamodel for M2 level. Although this is the easiest way and applied in practice, this solution sacrifices the reusability of the earlier work. The earlier models that are developed with the old M2-level metamodels will not conform to the new metamodel.

4.2.3. Extending an existing M2 layer metamodel

Another way is extending existing M2-level metamodels for proposing new domain specific metamodels for any stage. Every element of the new metamodel

should extend an element from the old metamodel, while conforming to the meta-metamodel of the old metamodel. In this case, the new metamodel will be called domain specific metamodel extension. Figure 4.1 presents the metamodel extension mechanism in MDD4MS.

The *extends* relationship between two metamodels expresses that: Metamodel-B *extends* Metamodel-A if each element in Metamodel-B extends an element from the Metamodel-A. In this way Metamodel-B is also an *instanceOf* of the meta-metamodel of Metamodel-A. DS-metamodel extensions include old concepts and new domain specific concepts. However, restrictions or constraints can be added such as writing rules for using only the new domain specific concepts.

Many state-of-the-art metamodeling environments provide automatic generation of visual model editors. Hence the extended metamodels can easily be used to generate a new modeling editor. The new editor can be customized while keeping the base structure of the old metamodel. The following sections present a detailed example of how to define a DS-metamodel extension for conceptual modeling stage of the MDD4MS lifecycle. First a generic simulation conceptual modeling metamodel is proposed and then it is extended to represent a function modeling language.

4.3. An extensible conceptual modeling metamodel for simulation

As stated in Section 2.1, there are many languages, techniques and tools for simulation conceptual modeling. But, there is not a commonly accepted conceptual modeling language for simulation. This is due to the fact that conceptual modeling languages are chosen to be understandable to the problem owner and so closer to the problem domain. Generic conceptual modeling languages have a potential risk of being insufficient for domain specific concepts, if they are not extensible and flexible. DSLs guarantee that the conceptual modelers work closer to the problem domain with a high level, intuitive and simple notation [TB11]. Metamodeling is one of the most practical and popular approaches for defining and using DSLs [AK03, AGRS13]. In this section, an extensible metamodel for a generic simulation conceptual modeling language is proposed. The language is mainly based on our earlier work presented in [ÇVS10b] and influenced from the systems theory.

4.3.1. SimCoML language and its metamodel

We propose a generic simulation conceptual modeling language (SimCoML) to define a system with its structure and abstract behavior. In SimCoML, a model consists of model parts. A model part can be a group modeling element, an atomic modeling element or an entity. Each part can have different types of variables such as input-output variables, state variables, parameters or model properties. Properties can define the meta information such as name, version, author, keywords, etc. They can be used in cataloging and searching services. Each part can have some constraints to express the boundaries of the system.

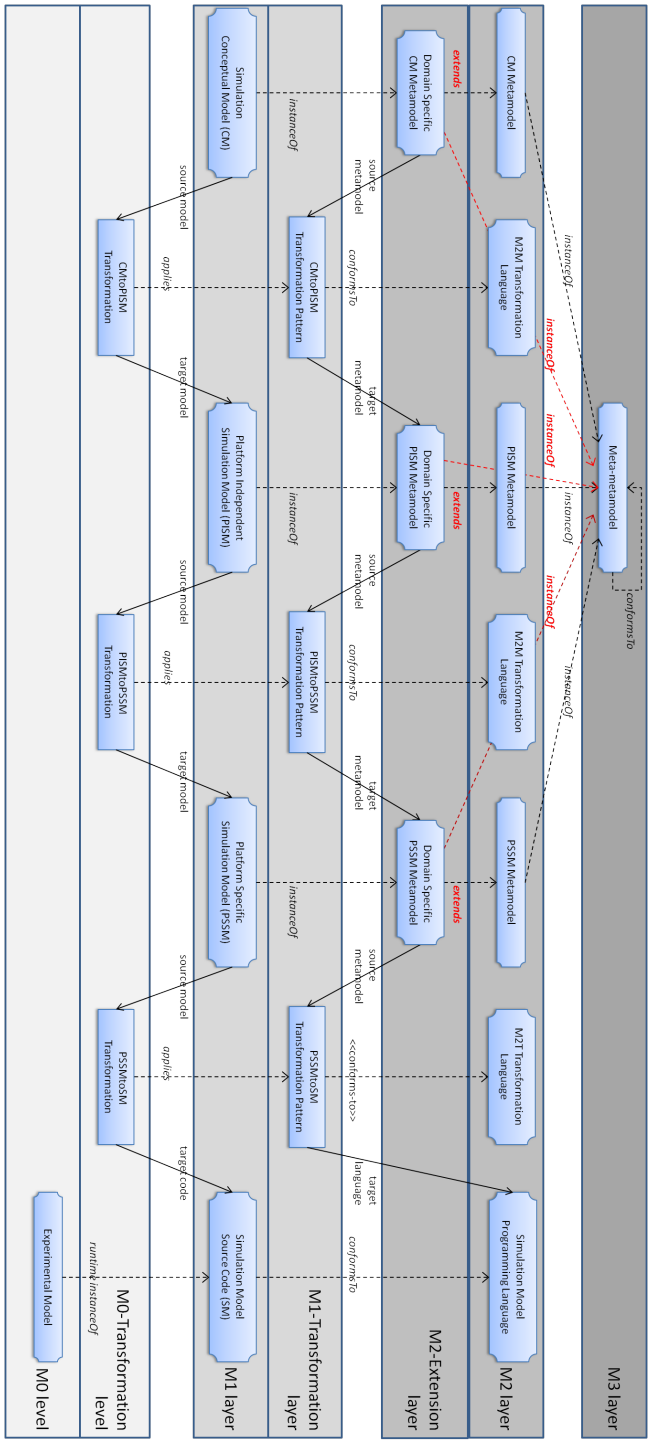


Figure 4.1: Domain specific metamodel extension mechanism in the MDD4MS framework.

Atomic model elements can include operations to represent system behavior. Variables and entities can be used in the definition of an operation. The total image of the state variables at a particular time shows the state of the system [LK91]. State change is possible when the state variables are updated. Output is available when the output variables are updated. Entities do not have operations and refer to the data types on the implementation level. For example, they can be used to represent system resources, objects, human actors, products, etc. Group model elements are used to organize the modeling parts in a hierarchical way.

Relations define how components and entities relate to each other. Four basic relations are proposed as fixed composition, inheritance, association and dynamic composition (aggregation). Fixed composition refers to the hierarchical structures in a model. Figure 4.2 shows the metamodel specified in Eclipse GEMS plugin for the SimCoML language.

In the metamodel, fixed compositions are represented with usual composition relation. Inheritance relation for entities is handled with BaseEntity attribute. BaseEntity can be specified if the entity inherits from an existing entity. Association relations or any other logical relationships between entities are expressed with LinkConnection. Appropriate cardinality information can be defined for the relations, such as: 1..*, *, 0..*, 0..1, 1, etc. The role of an entity in a relation can be defined as well. Aggregation, which refers to a temporary whole-part relationship, is represented with FlowConnection. A FlowConnection can carry a model part, whereas the default flow type is an 'entity' flow. This type of relation is called dynamic composition. EntityType for a FlowConnection can be either a single entity or a set of entities. A standard way of representation should be used since it needs to be parsed during the transformation. For example, '+' sign is used to represent a set in the next section such as 'Visitor + Ticket', which means that the visitor goes to the next step with a ticket.

In order to define the expressions a simple pseudo code mechanism is suggested as well. This mechanism can be used to support code generation. The expressions part of the metamodel is given in Figure 4.3. By using this metamodel a visual modeling editor is automatically generated. It is possible to add icons in the editor. Arrows show the flow of information.

4.3.2. A sample model for a queuing system

In order to provide a better understanding, a sample model for a single server queuing system is illustrated in this section. Simulation of a single server queuing system is a common example of discrete event simulation such as an information desk at an airport or a hotel, a pharmacy, a barber shop, or a ticket office.

For example, let's consider a service facility with a single server for which we would like to estimate the average waiting time in the queue for arriving customers. Let's say that it is a ticket office in front of a theme park. There will be three types of activities: arrivals, service and departure. The service activity is controlled with a

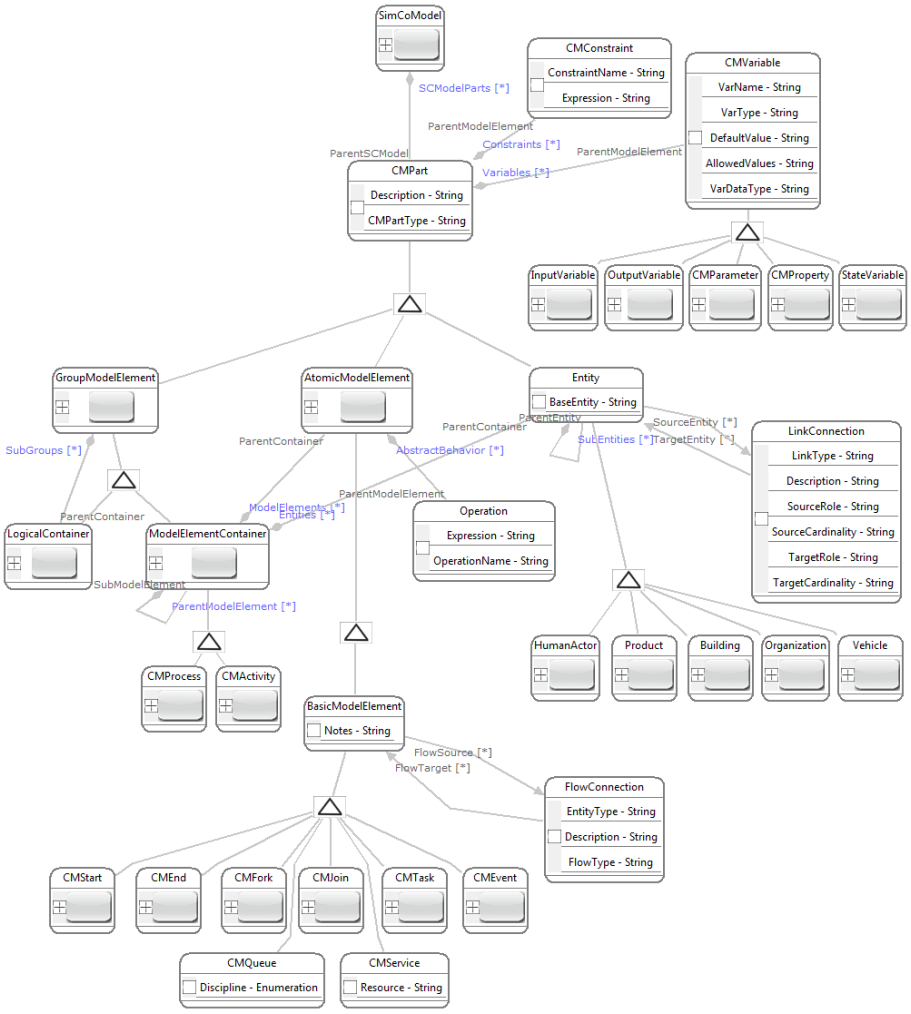


Figure 4.2: A metamodel for simulation conceptual modeling.

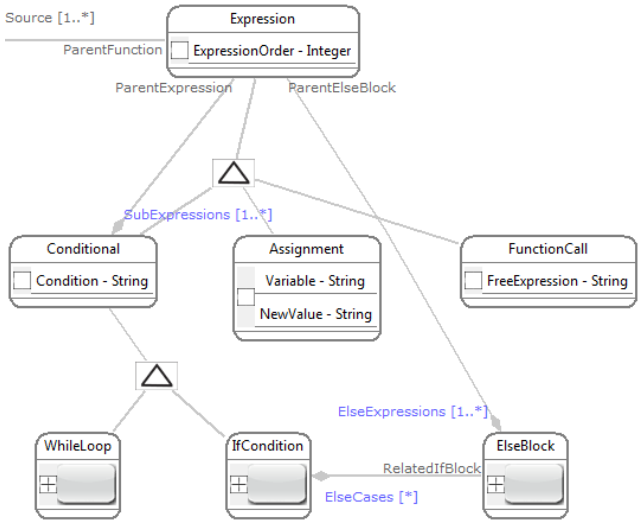


Figure 4.3: Pseudo code metamodel.

queue. The duration from the arrival of a customer to the queue until the instant he/she begins to be served will be calculated as the waiting time in the queue. The arrival and service patterns are expected to be specified in the formal model. A conceptual model for this system is illustrated in Figure 4.4.

4.3.3. Extending the metamodel for function modeling

A metamodel for IDEF0 is specified by extending the SimCoML metamodel and it is shown in Figure 4.5. IDEF (Integration DEFinition) is a family of modeling languages in the field of systems and software engineering [IDE99]. IDEF0 (Integration Definition for Function Modeling) is a function modeling method for describing functions of organizations or systems. An IDEF0 model consists of functions, data and objects. Functions are represented by boxes. Data or objects that interrelate those functions are represented by arrows. After extending the metamodel, a new IDEF0 editor is automatically generated which is still compatible with SimCoML models.

4.3.4. A sample model for order processing

A sample model that shows a ticket selling process is given in Figure 4.6. The process has four functions and is initiated with a customer order. The ticket seller accepts the order and then calculates the total price according to the price list. If the customer pays for the tickets, the ticket seller prints the tickets and gives them to the customer. Served customer leaves from the system.

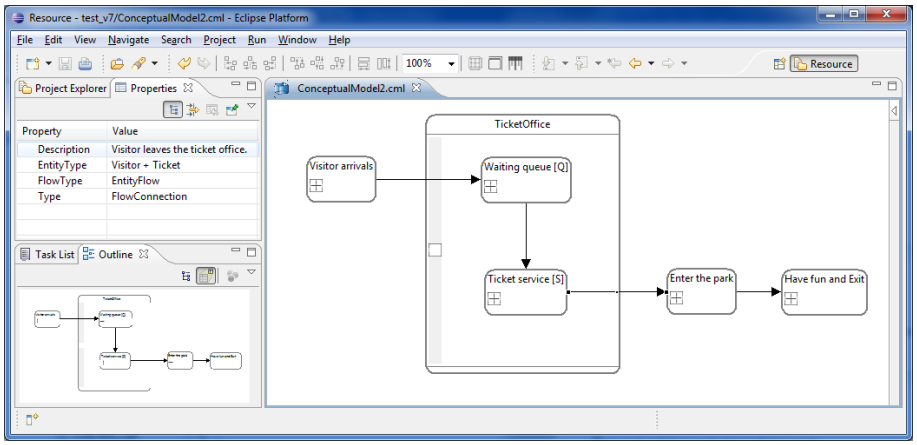


Figure 4.4: A conceptual model for a ticket office simulation.

4.4. How domain specific constructs support the model transformations?

In MDD, model transformations are not only expected to preserve the information in a source model but also enhance the model by adding new knowledge. In order to obtain an executable model or a big portion of the source code at the end, model transformations should add new knowledge at each step. If general purpose modeling languages are used in the earlier steps then it will be hard to generate specific data for the different parts of the model. But, if DSLs are used then each domain specific model element can be transformed into a more precise target element.

To discuss the positive effect of using domain specific constructs on model transformations, we provide two sample models in Figure 4.7. In the case of general purpose modeling language, although the modeler aims to model different types of activities in the model, he/she uses the generic activity element in the language. So, during the transformation it is not easy to handle the difference. For the model on the left hand side, same target code is generated for all of the four activities.

On the other hand, in the case of domain specific language, the difference is clear since the DSL already includes different constructs for various activity types. Hence, during the transformation, it is possible to generate more precise and useful target model elements or code. For the model on the right hand side, more detailed code including a queue mechanism is generated for a service activity whereas specific code is generated for printing and send message activities. Different icons help to distinguish different activities such as resource usage, printing or messaging. In this way, we expect to increase the usefulness of the model transformations.

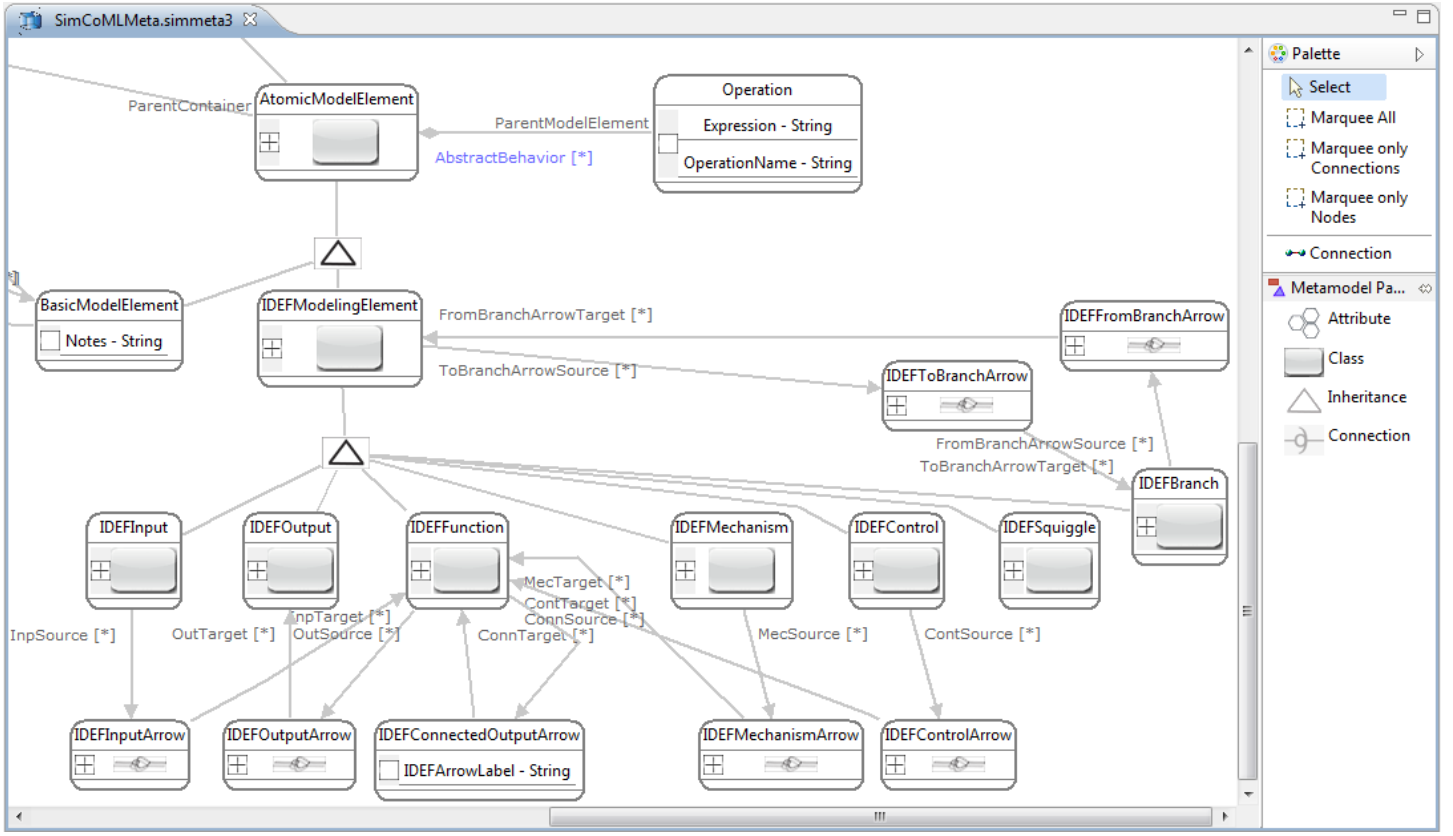


Figure 4.5: Extending SimCoML metamodel for IDEF0.

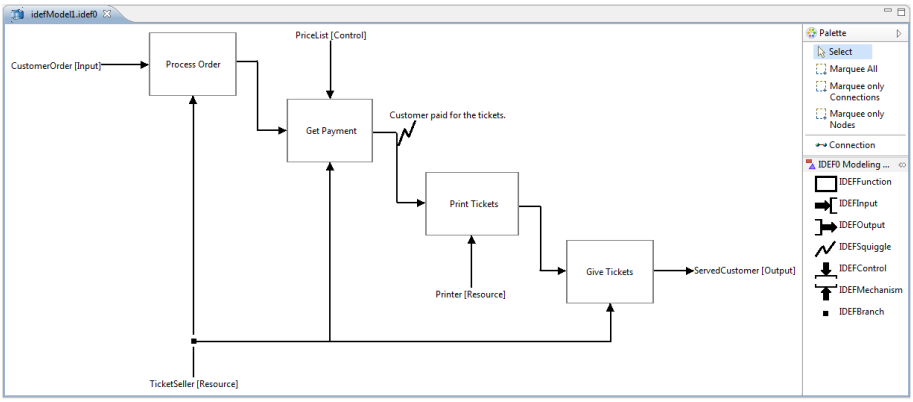


Figure 4.6: An example IDEF0 model for order processing.

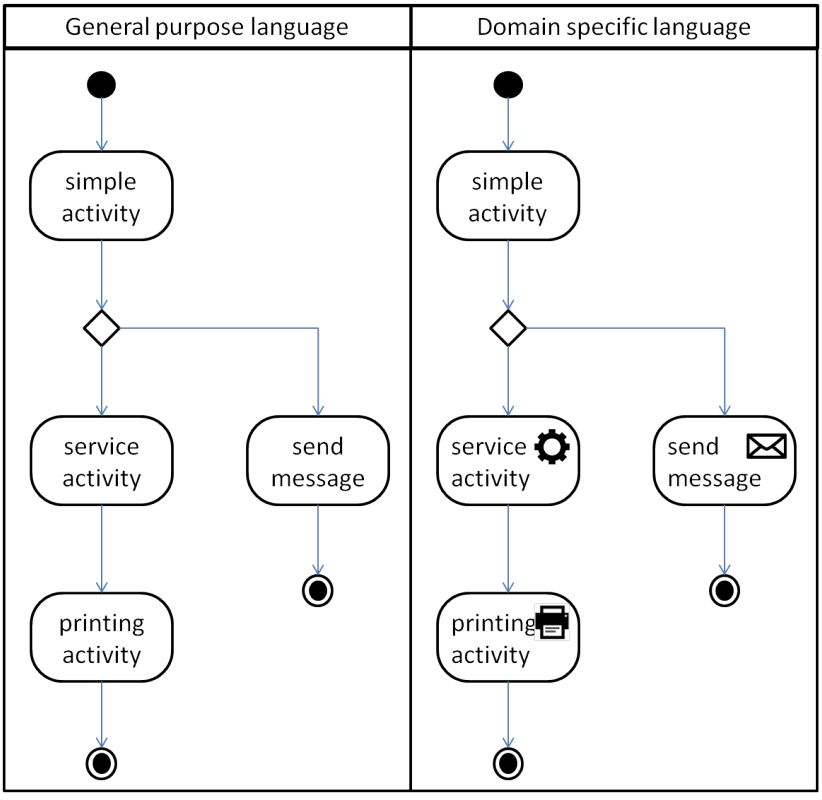


Figure 4.7: Domain specific constructs can be transformed into more precise target elements.

Chapter 5

Using Simulation Model Component Libraries

This chapter explains how to integrate simulation model component libraries into the MDD4MS framework in order to support model transformations. The full benefit of the MDD4MS framework can be better achieved if the component based approach is integrated in a systematic way. The component based approach has originally a bottom-up way of assembling components, and so requires a hierarchical modeling approach. Hence, the next section gives information about hierarchical modeling approach. After that, Section 5.2 presents information about component based simulation. Then, Section 5.3 explains how simulation model component libraries can be used in the MDD4MS framework.

5.1. Hierarchical modeling approach

Hierarchical modeling (also known as multi-level modeling) provides a way to represent a system in a hierarchical structure to deal with large scale or complex models in a thorough manner [Sim62]. Hierarchical modeling allows modeling with more manageable subparts at different levels of detail. The ability to move among the different levels of a model hierarchy greatly increases the manageability and understandability of large models [DS99]. As modelers build more complex and complicated models for large systems, thinking at various levels of abstraction becomes a useful approach. Hierarchical modeling can provide a more natural way of modeling and help to focus on various degrees of detail.

Hierarchical models are generally developed in two different ways as top-down or bottom-up strategies. In the top-down approach, a system is broken down into subsystems until the sub-subsystems are simple enough to be studied easily. This is called decomposition and once the simplest systems are developed, the composition of the subsystems produces the intended system. During the top-down modeling process, modelers specify the main parts and relationships of the system without inner details first and then they fill in the lower levels. The top-down approach can be especially useful when the details of lower level elements are not yet clear. In

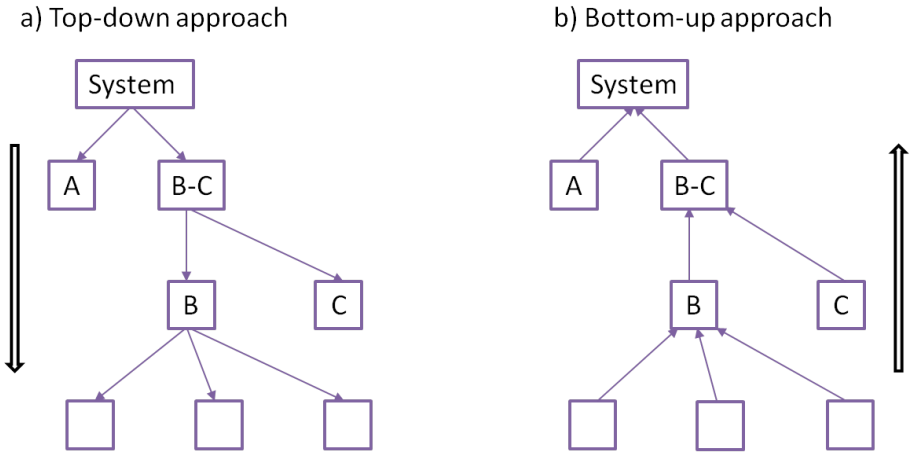


Figure 5.1: Hierarchical modeling approach.

the bottom-up approach, subsystems are coupled together to form a larger system and this is called composition. During the bottom-up modeling process, modelers first think of the lowest level, i.e. smallest parts or building blocks of the system and then they use these previously constructed building blocks to compose larger models and systems. The bottom-up approach has traditionally been used when pre-developed building blocks are available. Figure 5.1 illustrates the top-down and bottom-up approaches in hierarchical modeling. In both cases, hierarchical models generally represent a tree-like structure.

Simulation models can be developed by employing either a top-down decomposition approach or a bottom-up composition approach. Due to the fact that the simulation model development process starts with the system investigation and conceptual modeling activities, most of the M&S methodologies suggest a top-down modeling approach. For example, to represent an airport system, one would identify such subsystems as gates, security check points, information desk, check-in desks and so forth without delving yet into their inner details.

Due to some pre-packaged commercial tools offer low-level building blocks, the bottom-up approach is mainly adopted during the simulation model development in practice. Building block or component based simulation applies a bottom-up modeling approach. For example, agent based simulation is a common example of bottom-up approach where the agents are the fundamental building blocks of the model and the behavior of these agents produces the behavior of the system [Rob05, LDT13]. Figure 5.2 illustrates the top-down and bottom-up approaches in the M&S lifecycle. As a result, the theories do not match with the practice and so the component based simulation has not yet reached its potential. The next

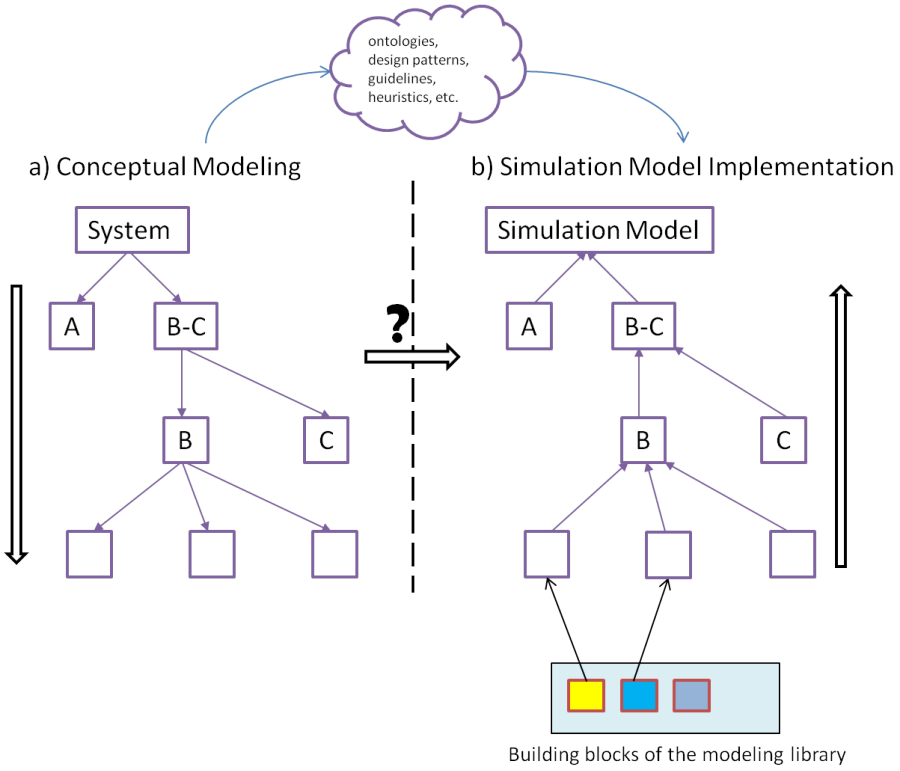


Figure 5.2: The top-down and bottom-up approaches in the M&S lifecycle.

section gives information about the component based simulation.

5.2. Component based simulation

In the early 1990s, component based approaches have emerged in software engineering with potential benefits for reduced development cost and time, effective use of subject matter expertise, increased quality through the reuse of certified artifacts and reduced risk [OPB04]. The component based approach promises reuse of interoperable components and rapid development. A software component is a unit of composition with precisely specified interfaces and explicit context dependencies. The development process for component based systems consists of two major stages: component development and component composition [OPB04]. These stages are usually carried out by different parties. When a component library is available, a developer can build a system in a bottom-up fashion, by combining components into larger components, where an assembly of the highest level components is considered to be the system. In component based approaches,

components are thoroughly tested first and reviewed during reuse, and so overall software quality increases [Som07]. However despite the benefits, it has many hidden risks such as lack of integration capability, lack of interoperability and lack of verification [DN06, Vit03].

Due to the fact that simulation models are becoming more and more complex and large, the development time and costs are increasing [OPB04]. Moreover it becomes hard to manage and maintain the simulation model after the development has been completed. The monolithic approach for developing models becomes too cumbersome in large simulation projects. When each simulation model is designed from scratch, the lack of reuse makes simulation a time consuming and expensive task. Applying the component based approach into the simulation field can help managing larger models [Bus00, HU04, SE09, VV08].

A simulation model component is expected to be a self-contained, interoperable, reusable and replaceable unit, providing useful services or functionality to its environment through properly defined interfaces [VD02]. Component based simulation (CBS) relies on having pre-built, validated simulation model components that can be coupled to form a hierarchical simulation model that represents a system. When applied successfully, this approach should significantly reduce the model development time. The component based approach has originally a bottom-up way of assembling components. Simulation model components can be assembled in many ways into a hierarchy. New components can be built from scratch in each layer or reused if they already exist in pre-defined and verified component libraries. A number of component based simulation frameworks have been proposed and many of them are developed mostly on a specific domain and for standalone use.

- VSE (Visual Simulation Environment) is an integrated development environment for creating and experimenting with discrete event, general-purpose, visual simulation models [BBEN98]. It allows creating libraries of reusable model components. VSE technology includes VSE Editor, VSE Simulator, VSE Output Analyzer, and VSE Teacher.

- Ptolemy II is another component based modeling and simulation framework developed, as a part of the Ptolemy Project [Pto07]. Ptolemy II uses a component specialization framework built on top of a Java compiler toolkit.

- CODES (COmposable Discrete-Event scalable Simulation) is systematic approach to component-based modeling and simulation that supports model reuse across multiple application domains [TS08]. A simulation component is viewed by the modeler as a black box. The attributes and behavior of the component are described using COML (COmponent Markup Language), a markup language proposed for representing simulation components.

- CoSMoS (Component Based System Modeling and Simulation) is an integrated modeling and simulation tool [SE09]. It has a unified concept

for specifying general-purpose logical, visual, and persistent primitive and composite models. Currently, CoSMoS supports developing parallel DEVS compliant models which can be executed using the DEVS-Suite simulator.

- Open Simulation Architecture (OSA) provides an open platform intended to support component-based discrete event simulation which is built on top of the Fractal component model [Dal06, RPMD09]. Fractal is the ObjectWeb Consortium component reference model, which is a set of rules and features that a component based software architecture is supposed to follow.

On the other hand, some pre-packaged commercial tools such as Arena, MATLAB-Simulink, Plant Simulation or Enterprise Dynamics offer solutions for limited domains and scopes. But, a clear metamodel and a proven formalism are lacking in many cases. In general, the components are not adaptable or extensible enough. They are not platform-free and not compatible with other components developed in different environments.

Although component based simulation looks like a promising field and its theory originated more than 20 years ago, many studies do not seem to deliver on the promises, and many simulation projects face problems when attempting to reuse existing components in practice. [KLV+10] presents a case study in which both simulation experts and novices experienced difficulties in developing a simulation model by using existing simulation building blocks. Novices and experts both utilized the building blocks to develop their models faster; however the models did not work as intended. The novices expected that they make mistakes and they started to make changes in their model. On the other side, the experts expected that the building blocks were incorrect and they tried to understand the logic and structure of the building blocks. Once they trusted the building blocks, they tried to change configuration of the building blocks in their model. At the end, they succeeded in correcting their models. The study showed that, the novices were more efficient in their design whereas the experts ran into a conflict between their own mental patterns and the design patterns offered to them, resulting in additional cognitive load. [Ver04] points to the difficulties in component based simulation as:

- Defining a complete, verified and consistent set of components is very difficult.
- Common M&S methodologies start from the assumption that models have to be built from scratch, and do not assume that there is a library of components available.
- Modelers are far from developing reusable components and reusing the predeveloped ones.
- Searching for the components, retrieving their functional and non-functional properties, understanding the conditions needed for successful operation, and

selecting the appropriate components can be more challenging than defining them from scratch.

Both component based software engineering and component based simulation can benefit from the successful implementation of component based approaches in other fields. For example, [YO06] provide a background for hardware assembly in engineering systems. In engineering applications, there are compatibility standards and each component is labeled accordingly. This type of labeling and documentation can be named semantic labeling and is very important for searching and selecting a component. Furthermore, a given hardware component may be interchangeable with a set of other components and this type of knowledge is also well documented. Besides, the design of the component is described with a hardware description language (HDL).

Therefore, similar considerations should be taken into account in simulation as well. Libraries and associated directory services should be provided for effective system composition. There must be a way to classify and distribute components efficiently. Good documentation is essential for the successful reuse of components. These are very important because otherwise developers will simply be unaware of what is already available. A set of simulation model components without any proper documentation about their usability, compatibility and interchangeability may not be useful and sufficient for successful practice of component-based simulation [YO06]. More information can be found in [Val11] about using simulation model components for effective simulation studies.

5.2.1. Component reuse

Reuse in M&S refers to the development of new models using pre-existing modeling elements like parts of a simulation code, functions, simulation components, and even similar simulation models. [Pid02] emphasizes four different types of model reuse as: code reuse (reusing or scavenging existing code), function reuse (reusing predefined functions that provide specific functionalities), component reuse (reusing encapsulated simulation modules that provide a well-defined interface) and full model reuse (reusing a pre-existing model).

Simulation models are typically built for individual projects and very little advantage is taken from existing simulation models developed earlier [KN00]. Due to the fact that existing M&S methodologies have no guidance for formal model transformations and many simulation projects have no deliberate conceptual modeling stage, simulation models generally do not have higher level representations, and so they are not easily understandable by others. Thus, many redundant representations of the same concepts are developed in M&S projects. In large scale projects, lack of reuse makes simulation a time consuming and expensive task [KN00].

Introducing a component based approach into the simulation field can help with the reuse problem, since it promises reuse of interoperable components and hierarchical

modeling [Ver04, Val11]. Components can be assembled in many ways into a hierarchy and an assembly of the highest level components is considered to be the simulation model. Reuse of simulation model components has the potential of reducing the cost of specification, coding, documentation, maintenance, validation and verification [BBEN97]. Using a component based design approach in the earlier stages is also desirable to bridge the gap between the conceptual modeling and the simulation modeling stages [BvET01, BJT02].

In the real life, reuse is more likely to occur in reusing existing code and functions. Unfortunately, code reuse is more likely the copy, paste and change operation, which is not considered true reuse in software engineering. True reuse requires an instantiation capability [BBEN98]. Component reuse provides the possibility of reusing pre-developed components (or building blocks) and refers to utilizing domain specific component libraries [Ver04]. On the other hand, reusing a full model without any modification is feasible only if we intend to solve exactly the same problem or subproblem that the model was intended for [BAA08]. In most of the cases, however, we neither model the same system nor do we intend to solve the very same problem. Reusing a model for a purpose other than for which it was originally constructed requires modifications on the model itself. Although component reuse and model reuse has been a goal for a long time, they have not been achieved effectively [Pid02]. Thus, the developed simulation model components are rarely reused and often not used after the first simulation study. The main problem in component based simulation is model composability such that the simulation model components are usually platform dependent and not compatible with other components developed in different environments [KN00, RU06, YO06].

There are many different research fields that have a direct or indirect effect on simulation model component reuse such as collaborative problem solving, using patterns in modeling, open source programming, using reusable assets for packaging, using metadata or object data information etc. In this thesis, the approach is about using platform independent model templates for providing a way to express how the implemented components relate to the subtrees in the formal model.

5.2.2. Requirements for simulation model components

This section briefly describes some basic requirements for simulation model components derived from [VV08] and [OPB04]. A successful practice of component based simulation can be performed by fulfilling these requirements.

- **R-CBS.1** Modularity: Components must be self contained and modular. Internal data should be used within a component; and external data and processes must be used through the interfaces.
- **R-CBS.2** Interoperability: Components must be interoperable, i.e. it must be capable of working together with other components. This requirement includes assembly of components in different layers.

- **R-CBS.3** Reusability: It must be possible to reuse a component by instantiating it in different simulation models.
- **R-CBS.4** Functionality: Components must provide useful functionality which is the role, service, operation, or whatever the component provides to the overall system.
- **R-CBS.5** Upgradeability: It should be possible to upgrade a component with the extended version. The effects of upgrades in the component libraries should be unambiguous.
- **R-CBS.6** Replaceability: It should be possible to replace a component with another component with the same interfaces and similar function.
- **R-CBS.7** Reachability: Components must be easily reachable. They should be classified, categorized and kept in searchable libraries.
- **R-CBS.8** Flexibility: It must be easy to configure the component by using its parameters and to extend its capabilities.

5.3. Using simulation model component libraries within the MDD4MS framework

The MDD4MS framework highly motivates and supports component based simulation. A library based approach is recommended to make component use convenient and practical for the modelers via well documented simulation model component libraries. These libraries contain pre-built, parameterized and flexible domain specific components for reuse. The validation and verification of components are significant for successful reuse. Standardization and certification of components are also important for ensuring trust and promoting general use. Simulation model component libraries can be developed for particular application domains in the M&S field such as health care, transportation, logistics or airport systems.

In order to support the model transformations within the MDD4MS framework, a simulation model component is suggested to have an associated formal model and an implementation model. The formal model of the component will be called a PISM model template. The implementation model will be called a PSSM model template. By using the model templates formal PISMs and PSSMs can be developed. During the PISMtoPSSM model transformation, PISM model templates are replaced with the PSSM model templates. Similarly, during the PSSMtoCode model transformation, PSSM model templates are replaced with the instances of the linked components. Although, one can think about CM model templates, it is out of the context of this thesis. However, during the CMtoPISM model transformation, some parts of the conceptual model can be transformed into PISM model templates according to some pre-determined assumptions.

In short, a model template is a pre-developed and archived model on M1 level. It is expected to be parametric and have an associated simulation model component.

A set of model templates for a specific domain with the associated component information and good documentation forms a domain specific template library for MDD4MS. In this case, the set of modeling elements provided by the grammar of the formal specification language may be called the basic (or core) modeling library [AK02]. Figure 5.3 presents the model template mechanism in MDD4MS. Each element in the template has an *instanceOf* relationship to the higher level metamodel. When a template library is available, the notion of *includes* means that the model uses a template from a library and so reuses a component. The PISM templates refer to PSSM templates and PSSM templates refer to components. The model transformation patterns use templates and components.

The conceptual modeling and specification layers are related through a mapping relation. An element in a conceptual model can be mapped to an element in the PISM library which delivers the desired functionality. This feature offers continuity between the conceptualization and specification phases with simpler transformation rules. In the same way, specification and implementation are related through a matching relation. Any part of a PISM model can be matched to a PSSM model template. For each formal specification, a number of platform specific implementations can be developed. For example, a formal DEVS model can be implemented using various tools and platforms. Although in the ideal case a simulation model component will have both PSSM template and a PISM template which are clearly linked, it is also possible to have not implemented PISM templates to provide only formal specification.

Figure 5.4 proposes a sample workflow for the proposed model template mechanism. In this workflow, it is assumed that transformation rules are overridden if model templates are used. During the specification stage the modeler searches for the existing components according to their PISM model templates and selects the most appropriate ones. A mapping between the CM elements and the PISM templates is done. During the implementation stage the modeler searches for the existing components according to their PSSM model templates and selects the most appropriate ones. A mapping between the PISM elements and the PSSM templates is done. If the PISM template has an already linked implementation then it should be preferred. Then, the programmer defines and develops the missing components and adds them to the library.

The most important benefit of using template libraries is that the model transformation rules will be simpler. Instead of transforming a concept from the source metamodel to a target concept with all parts and details, only a link to the related model template is defined. This link should include the all required information to be able to reach the exact component. By using domain specific component libraries it is possible to generate a fully executable simulation model from a conceptual model based on the assumptions and abstractions made in the components.

One of the discussed issues of MDD is the code redundancy problem, which comes with the automatic code generation techniques. Since the code is auto generated

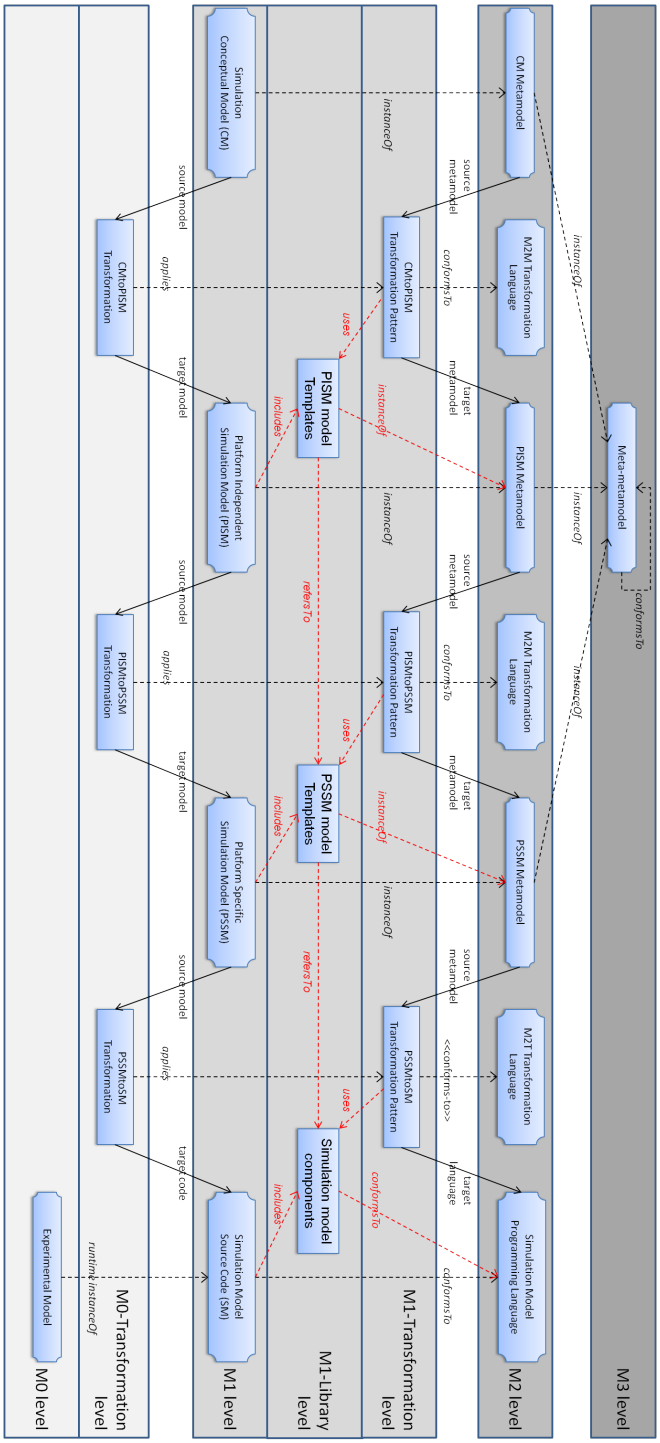


Figure 5.3: The model template mechanism in MDD4MS.

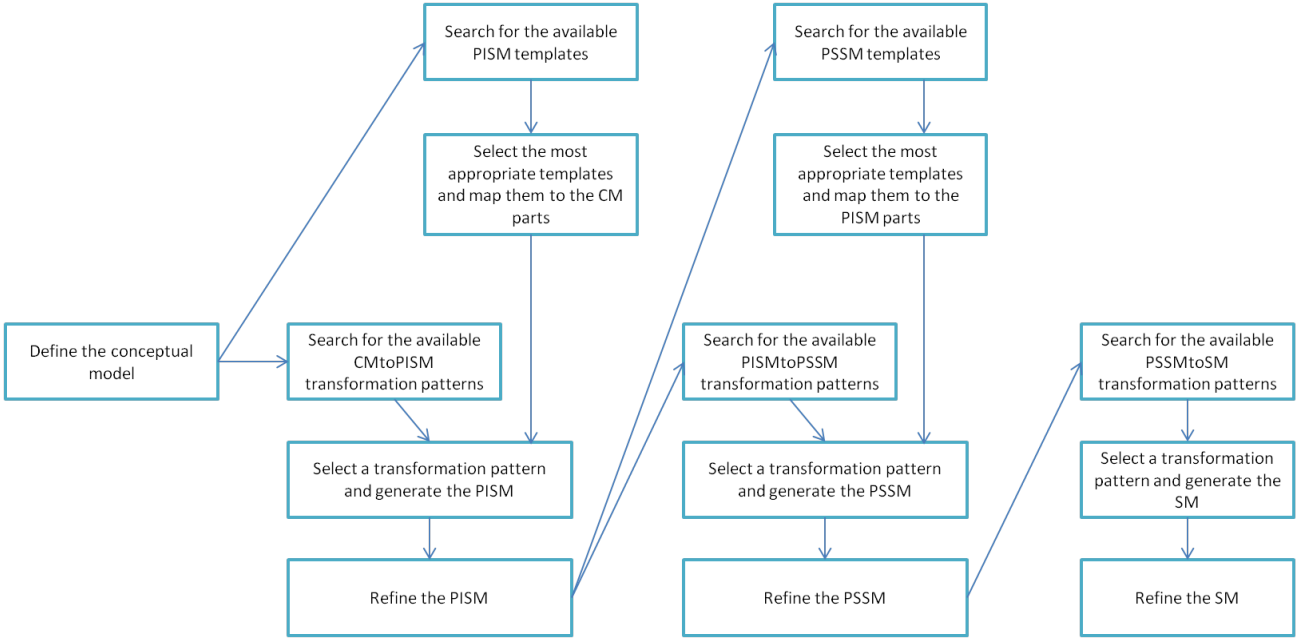


Figure 5.4: A sample workflow for the model template mechanism in MDD4MS.

there is sometimes redundant and repetitive code in the final software. Using simulation model component libraries provides a solution for this problem. Instead of generating new code during the PSSMtoCode transformation, existing components are reused.

For example, assume that for each element with name 'task_*' in a PSSM model, JAVA code will be generated and the generated classes will be instantiated. A sample model includes task_1, task_2 and task_3. Without any component based approach or any code optimization effort, a model transformation can generate task_1.java, task_2.java and task_3.java, which will have same structure. Naturally, in this way, all the transformation details are kept in the model transformation, which makes them more complex. A potential instantiation of these classes will become as below:

```
var t1 = new task_1();
var t2 = new task_2();
var t3 = new task_3();
```

On the other hand, if task.java is written before and saved as a template, then the model transformation can use the meta-information of this class. With an available component, model transformation will not regenerate the code but use the pre-developed code. This will incredibly decrease the code redundancy. Besides, the transformation rules will be shorter. A potential use of the task class will become as below:

```
var t1 = new task( 'task_1');
var t2 = new task( 'task_2');
var t3 = new task( 'task_3');
```

Chapter 6

Case: Discrete Event Simulation of Business Process Models

This chapter presents an application of the MDD4MS framework with a proof of concept implementation. The study is about discrete event simulation of business process models. Hence, the following two sections present information about business process modeling and discrete event simulation. After that the implementation and the MDD4MS prototype is explained in Section 6.3. Then, the case study is presented with two examples and it is evaluated in the next sections. Lastly the results are analyzed in Section 6.6.

6.1. Business process modeling

Business process modeling (BPM) is the activity of defining a graphical representation of either the current or the future processes of an organization in order to analyze and improve their efficiency and quality [RRIG09]. A business process model is a visual representation of a set of related activities. There are numerous methods and tools available for BPM and some of the tools provide ways to simulate models. Business process simulation (BPS) enables analysis of business processes over time and allows to experiment with what-if scenarios before implementing the ideas into the organization. BPM is also known as static modeling while BPS is known as dynamic modeling [BVCH07].

BPS tools are used for the evaluation of the dynamic behavior of business processes. Different domain-specific languages have been used to develop business process models [RRIG09]. IDEF and BPMN are the most common BPM techniques, and BPMN [OMG11a] is selected in this case study. BPMN is an industry-wide standard for modeling of business processes [OMG11a]. BPMN follows the tradition of flowcharting notations for readability and flexibility. In addition, the BPMN execution semantics is fully formalized. A BPMN model provides a high-level representation of a business process that is readily understandable by business analysts and technical developers.

There are five basic categories of elements in BPMN. These are flow objects,

data elements, connecting objects, swimlanes and artifacts. Flow objects are the main graphical elements that define the behavior of a business process. There are three types of flow objects: events, activities and gateways. Data elements are represented by the four following types: data object, data input, data output and data store. There are four ways of connecting the flow objects to each other or to other elements: sequence flows, message flows, associations, and data associations. There are two ways of grouping the primary modeling elements through swimlanes: pools and lanes. Artifacts are used to provide additional information about the process. BPMN also defines the visual representation of each element.

6.2. Discrete event simulation

Discrete event simulation (DES) is a popular and effective method for analyzing and designing systems. It has been successfully used in many different areas such as transportation, manufacturing, construction, telecommunications, military, and health care.

State and time descriptions form the basis of a specification formalism. In [Nan81], the state of an object is defined as the record of all attribute values of that object at a particular time; and time is defined as an attribute of the model that enables state transitions. Two common measures of time are instant and interval. An instant is a value of system time at which the value of at least one attribute of an object can be altered, i.e. it is a point in time. An interval is the duration between two instants.

In a discrete event system, the system's state changes at discrete points in time upon the occurrence of an event. Discrete-event simulation assumes that, although time is continuous, only a finite number of events can occur in a given finite time interval. Therefore, the execution of a discrete-event simulation model can be very efficient because it is only needed to represent the state changes upon occurrence of events [Wai09]. Simulation of a single server queuing system is a very common example of DES.

There are three common simulation model development perspectives for DES, i.e. world views: event scheduling, activity scanning, and process interaction. The time and state concepts can be used to define the event, activity and process terms [Nan81]. An event is a change in object state, occurring at an instant that initiates an activity. An activity is the state of an object over an interval. A process is the succession of states of an object over an interval of two successive instants. In event scheduling, each event routine in a model specification describes related actions that should always all occur in one instant (Time based). In activity scanning, each activity routine in a model specification describes all actions that should occur due to the model assuming a particular state (State based). In process interaction, each process routine in a model specification describes the action sequence of a particular model object (Object based) [ON04].

Process interaction world view is basically a combined event scheduling and activity scanning procedure. The distinguishing feature is that a model component description can be implemented as a unit rather than being separated into a number of events or activity routines. Process interaction simulations are typically implemented on top of event driven simulation mechanisms [Fuj00]. Specifically, process interaction simulations use the same event list and time advance mechanism defined for the event driven world view but provide additional mechanisms for managing simulation processes. Process interaction simulations often utilize the concept of a resource, which is an abstraction that represents a shared entity. Overstreet and Nance [ON04] present characterizations and relationships of world views and illustrate the possibility of automated transformation among world views.

Discrete Event System Specification (DEVS) is a well known mathematical formalism based on system theoretic principles [ZPK00]. Mathematical systems theory, first developed in the 1960s, provides a fundamental mathematical formalism for representing systems [Wym67]. It is concerned with the dynamic behavior of the system and the state changes over time.

Zeigler et al. [ZPK00] claim and show that any system with discrete event behavior can be represented with the DEVS formalism and an equivalent DEVS representation can be found for other formalisms. In other words, being a general system theoretic formalism, DEVS allows one to represent all the systems that have discrete event behavior. This means that discrete event systems modeled by different techniques such as Petri Nets, State Charts, Partial Differential Equations, Bond Graphs, Finite State Automata, etc. can be represented by DEVS models. Discrete time systems can be represented by DEVS as well [ZPK00]. The generality of DEVS converted it into a widely studied formalism and many DEVS based modeling and simulation environments have been developed in recent years to describe and to simulate many classes of discrete systems [SE09]. Hence, DEVS is selected as the system specification formalism.

In DEVS, models that are expressed in the basic formalism are called atomic models. Hierarchical DEVS is the extended version of the basic formalism that defines the means for coupling the DEVS models. The composite models are called coupled models. An atomic DEVS model is defined with the following information: the set of input values, the set of output values, the set of state variables, the internal transition function, the external transition function, the output function and the time advance function. Functions define the system dynamics.

DEVS uses named input and output ports to symbolize the connection points between the models and to provide an elegant way of building composite models. Larger models are built by coupling models in a coupling scheme that links the input ports and the output ports. If the couplings are done correctly, the resulting coupled model is regarded as closed under coupling, which means that it can be expressed as an atomic DEVS model.

In DEVS, atomic components have state but coupled components have a derived state, which is in fact the set of the state of the composed atomic components. A coupled DEVS model is defined with the following information: the set of input ports and values, the set of output ports and values, the set of the components and the couplings. Components are DEVS models and couplings can be EIC(external input coupling that connects external inputs to component inputs), EOC (external output coupling that connect component outputs to external outputs) and IC (internal coupling that connects component outputs to component inputs). Coupled DEVS models allow constructing hierarchical models by using components' external interface provided with ports. It should be noted that each DEVS model is self contained and executable.

An atomic DEVS model is defined in [ZPK00] as below:

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

where

X is the set of input values,

S is a set of states,

Y is the set of output values

$\delta_{int} : S \to S$ is the *internal transition* function

$\delta_{ext} : Q \times X \to S$ is the *external transition* function, where

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the *total state* set

e is the *time elapsed* since last transition

$\lambda : S \to Y$ is the output function

$ta : S \to R^{+}_{0,\infty}$ is the set of positive reals with 0 and ∞

At any time the system is in some state, *s*. If no external event occurs, the system will stay in state *s* for time *ta(s)*. When the elapsed time $e = ta(s)$, the system outputs the value $\lambda(s)$ and changes to state $\delta_{int}(s)$. Output is only available just before internal transitions. If an external event $x \in X$ occurs when the system is in total state *(s,e)* with $e \leq ta(s)$, the system changes to state $\delta_{ext}(s, e, x)$. A coupled DEVS model is defined in [ZPK00] as following:

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select)$$

where

$X = \{(p,v)|p \in IPorts, v \in X_p\}$ is the set of input ports and values,

$Y = \{(p,v)|p \in OPorts, v \in Y_p\}$ is the set of output ports and values,

D is the set of the component names. Components are DEVS models, for each $d \in D$,

$M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ is a DEVS with

$X_d = \{(p,v)|p \in IPorts_d, v \in X_p\}$

$Y_d = \{(p,v)|p \in OPorts_d, v \in Y_p\}$

$EIC \subseteq \{((N, ip_N), (d, ip_d))|ip_N \in IPorts, d \in D, ip_d \in IPorts_d\}$ external input coupling connect external inputs to component inputs

$EOC \subseteq \{((d, op_d), (N, op_N))|op_N \in OPorts, d \in D, op_d \in OPorts_d\}$ external output coupling connect component outputs to external outputs

$IC \subseteq \{((a, op_a), (b, ip_b))|a, b \in D, op_a \in OPorts_a, ip_b \in IPorts_b\}$ internal coupling connects component outputs to component inputs

(However, no direct feedback loops are allowed, i.e. no output port of a component may be connected to an input port of the same component:

$((d, op_d), (e, ip_d)) \in IC$ implies $d \neq e$.)

$Select : 2^D - \{\} \rightarrow D$, the tie-breaking function

6.3. MDD4MS prototype implementation

In Chapter 3, the MDD4MS framework is proposed for model-driven development of simulation models through metamodel based model transformations. A tool architecture for the framework is proposed in Section 3.5. The MDD4MS prototype is an Eclipse-based implementation of the tool architecture. The Eclipse platform is chosen since it is designed for building new integrated development environments that can be used to create applications. Besides, Eclipse community provides plugins for applying MDD approach and creating graphical editors. During the implementation, we preferred open source and flexible tools since this is a research project and we could require improving the tools according to new research needs.

The prototype is based on the Eclipse Modeling Project and the Eclipse Modeling Framework (EMF) [Ecl09]. All of the tools that have been used are the subprojects of the top-level Eclipse Modeling Project. EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XML Metadata Interchange (XMI), EMF provides tools and run-time support to produce a set of Java classes for the model, along with a set of adapter classes that enable view-

ing and command-based editing of the model, and a basic editor. The core EMF framework includes a meta-metamodel (Ecore) for describing models and run-time support for the models, including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects.

6.3.1. Metamodeling with the GEMS Project

The Eclipse-based Generative Modeling Technologies (GMT) project provides a set of prototypes and research tools in the area of MDD [Ecl06]. Historically, the most important operation was model transformation, but other model management facilities, like model composition, are also part of the GMT project. Different subprojects are proposed in the GMT project. The Generic Eclipse Modeling System (GEMS) in the GMT project is a configurable tool kit for creating domain-specific modeling and program synthesis environments for Eclipse. The GEMS project provides a visual metamodeling environment based on EMF and GEF/Draw2D. It includes a code generation framework in which a graphical modeling editor is generated automatically from a visual metamodel specification. The graphical modeling editor can be used for editing instances of the modeling language described by the metamodel. Although the generated editor uses a default concrete syntax, the graphical representation of the modeling elements and the visual appearance of the editor can be customized by changing a CSS style sheet file. The generated graphical modeling tool is based on EMF, GEF, and Draw2D and it can be exported as an Eclipse plug-in. Besides, it can be extended through extension points for writing model interpreters.

The built-in metamodeling language is based on the UML class diagram notation. Metamodels are directly transformed into Ecore meta-metamodel. Metamodels in other Ecore readable formats can be used as well. The models defined by the generated tools are saved as XMI files. Metamodel constraints can be specified in Java. GEMS project provides a meta-programmable modeling environment similar to GME for the Eclipse community and it is open source. The MDD4MS prototype is based on GEMS with some small improvements for increasing the graphical modeling capabilities.

6.3.2. M2M transformations with ATL

Model-to-model (M2M) transformation is a key aspect of MDD. The Eclipse M2M project presents a framework for defining and using model-to-model transformation languages. The core part is the transformation infrastructure. Transformations are executed by transformation engines that are plugged into the infrastructure. There are three transformation engines that are developed in the scope of Eclipse M2M project, which are ATLAS Transformation Language (ATL), Procedural Query/View/Transformation (QVT) (Operational), and Declarative QVT (Core and Relational). Each of the three represents a different category, which validates the functionality of the infrastructure from multiple contexts. The ATL IDE

aims to ease the development and execution of ATL transformations [JABK08]. The MDD4MS prototype uses the ATL IDE for the M2M transformations in the MDD4MS framework.

6.3.3. M2T transformations with visitor-based model interpreters

The GEMS project has a visitor-based model interpretation mechanism . For each modeling element specified in the metamodel, a corresponding visitor method is generated in the interface. A Java project with a class that implements these methods can be developed as a model interpreter. When the model interpreter is executed, each element in a particular model can be visited separately and the required code can be generated. The model interpreters are registered to the editor by adding an extension point to the MANIFEST.MF file in the META-INF directory. The current MDD4MS prototype makes use of this model interpretation mechanism.

6.4. DEVS-based simulation of BPMN models

This section explains the conceptual application of the MDD4MS framework for DEVS-based simulation of BPMN models. The DEVS-based simulation of BPMN models requires a transformation process from BPMN to DEVS. The modeling elements of BPMN have to be mapped to DEVS components to be able to simulate their behavior in a DEVS simulation environment. In our case, composed elements such as pools and lanes are mapped to Coupled-DEVS, whereas other basic elements such as events and gateways are mapped to Atomic-DEVS. Flow connections are expressed with coupling relations in coupled models. Input and output ports are defined for each component. A supplementary component called Resource Manager is designed to support the simulation functionality for resource allocation and waiting queues.

DSOL (Distributed Simulation Object Library) is selected to provide the simulation and execution functionalities [JLV02]. DSOL is an open source multi-formalism simulation suite which is full featured and very effective as a generic purpose simulation tool. Various simulation projects have been conducted using DSOL in a broad range of application areas, including flight scheduling, emulation and web-based supply chain simulation [JVM05, KV07, FSV10, HSV10]. DSOL also supports execution of simulation models based on the DEVS formalism through the DEVSDSOL library [SV09]. DEVSDSOL provides an object-oriented conceptualization of DEVS language constructs and implements a DEVS compliant modeling and simulation environment using the event-scheduling worldview. DSOL and DEVSDSOL are both written in the Java programming language.

DEVSDSOL defines AtomicModel and CoupledModel abstract classes. Building an atomic model is done by extending the abstract class AtomicModel, instantiating input and output ports, creating state variables and phases and overriding the abstract methods specifying the DEVS functions (deltaExternal, deltaInternal,

lambda and timeAdvance functions). Similarly, building a coupled model is done by extending the abstract CoupledModel class, adding input and output ports, adding model components, and defining the connections within the coupled model. InputPort and OutputPort are defined as member classes of AtomicModel and CoupledModel. As a result,

- BPMN is chosen as the conceptual modeling language to define conceptual models,
- DEVS is chosen as the system specification formalism to define platform-independent simulation models,
- Java and the DEVSDSOL library in particular, are chosen as the underlying simulation model programming languages to define platform-specific simulation models.

Following the MDD4MS framework, the following metamodels and model transformation are required during the application of the MDD4MS framework:

- BPMN metamodel as the CMmetamodel,
- DEVS metamodel as the PISMmetamodel,
- JAVA metamodel as the PSSMmetamodel,
- BPMNtoDEVS transformation as the CMtoPISM transformation,
- DEVStoJAVA transformation as the PISMtoPSSM transformation,
- JAVAtoJAVACode transformation as the PSSMtoCode transformation.

As a result, we formally define our case as follows:

Definition 18 (MDD4MS case study)**.** *Model driven development of DEVS-based simulation models from BPMN models is an MDD4MS process defined as*

$$case = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, SM, TO\}$$

where

$n = 3$ $(CM, PISM, PSSM)$*,*

$MML = \{Ecore, Ecore, Ecore\}$ *is the ordered set of metamodeling languages,*

$ML = \{l_0(BPMNmetamodel), l_1(DEVSmetamodel), l_2(JAVAmetamodel)\}$ *such that*

$\gamma(BPMNmetamodel, Ecore)$*,*

$\gamma(DEVSmetamodel, Ecore)$,

$\gamma(JAVAmetamodel, Ecore)$,

$MO = \{CM, PISM, PSSM\}$ *such that CM is the initial model, PSSM is the final model, and*

$\tau(CM) = BPMNmetamodel$,

$\tau(PISM) = DEVSmetamodel$,

$\tau(PSSM) = JAVAmetamodel$,

$SL = \{ATL, JAVA\}$ *is the set of model transformation languages,*

$pl = JAVA$ *is the programming language and it is extended with the DSOL and DEVSDSOL simulation libraries,*

$MTP = \{p_{cm}, p_{pism}, p_{pssm}\}$ *such that*

$p_{cm} = \{l_0(BPMNmetamodel), l_1(DEVSmetamodel), bpmn2devs.atl\}$,

$p_{pism} = \{l_1(DEVSmetamodel), l_2(JAVAmetamodel), devs2java.atl\}$,

$p_{pssm} = \{l_2(JAVAmetamodel), JAVA, java2code.java\}$,

$STP = \{\}$ *is the set of other supplementary formal model transformation patterns,*

$MT = \{(\theta(CM, p_{cm}) = PISM)$, $(\theta(PISM, p_{pism}) = PSSM)$,

$(\theta(PSSM, p_{pssm}) = SM)\}$,

SM *is the final executable simulation model,*

$TO = \{$*Eclipse and a set of plugins*$(GEMS, ATL, PDE, EMF, GEF)\}$ *is the set of tools to ease the activities.*

The overall MDD4MS architecture, which is presented in Figure 3.2, is instantiated as shown in Figure 6.1. Figure 6.2 shows the application of the MDD4MS process for DEVS-based BPMN simulation.

6.5. Practical implementation with the MDD4MS prototype

This section explains the implementation of the MDD4MS process for the case study with the MDD4MS prototype. The implementation includes metamodels, model editors, model transformation rules, and model interpreters for DEVS-based simulation of BPMN models [ÇVS13, ÇVS12, ÇVS11]. The following sections explain the details of the implementation.

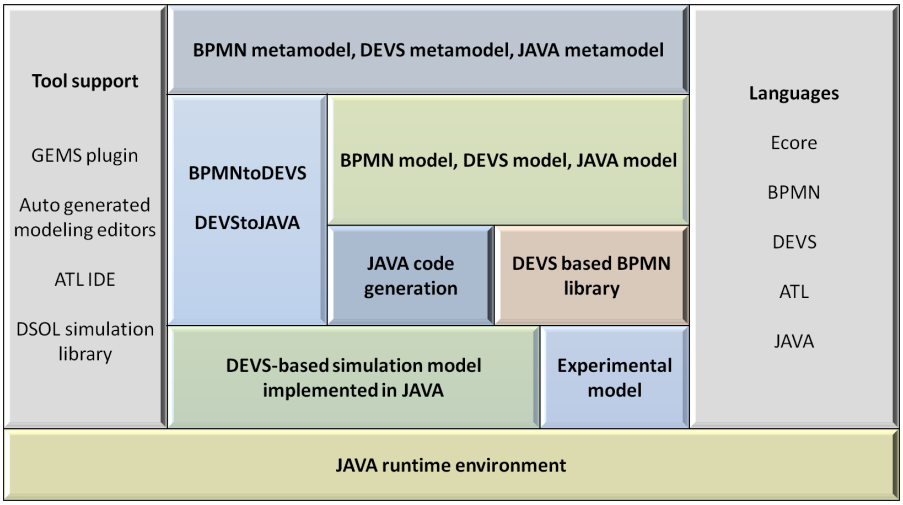


Figure 6.1: The overview of the case study.

6.5.1. BPMN metamodel

The BPMN metamodel is defined with the GEMS plugin, and a modeling editor and an Ecore metamodel are automatically generated. The metamodel is shown in Figure 6.3. BPMNDiagram represents the business process model. The main graphical element of the diagram is BPMNFlowObject. BPMNEvent, BPMNActivity, and BPMNGateway inherit from the flow object. An event is something that happens during the course of a process. There are three types of events, based on when they affect the flow: start event, intermediate event, and end event. An activity is a task performed in a process. An activity can be an atomic task or a compound subprocess. A gateway is used to control the divergence and convergence of the sequence flows in a process. Thus, it determines the branching, forking, merging, and joining of paths. Internal markers indicate the type of behavior control. Each type of control affects both the incoming and outgoing flow. BPMNParallelFork and BPMNParallelJoin are defined for parallel forking and joining; and BPMNDecide and BPMNMerge are defined for exclusive decision and merging.

BPMNTokenFlowConnection represents the flow between the flow objects. The type of flow is determined by the FlowType attribute. The sequence flow is used to control the order of the activities in a process. Swimlanes are used to group activities. A pool is used as a graphical container for partitioning a set of activities. A lane is a subpartition within a process, sometimes within a pool, and will extend the entire length of the process, either vertically or horizontally. Lanes are used to organize and categorize activities. The metamodel represents the basic modeling

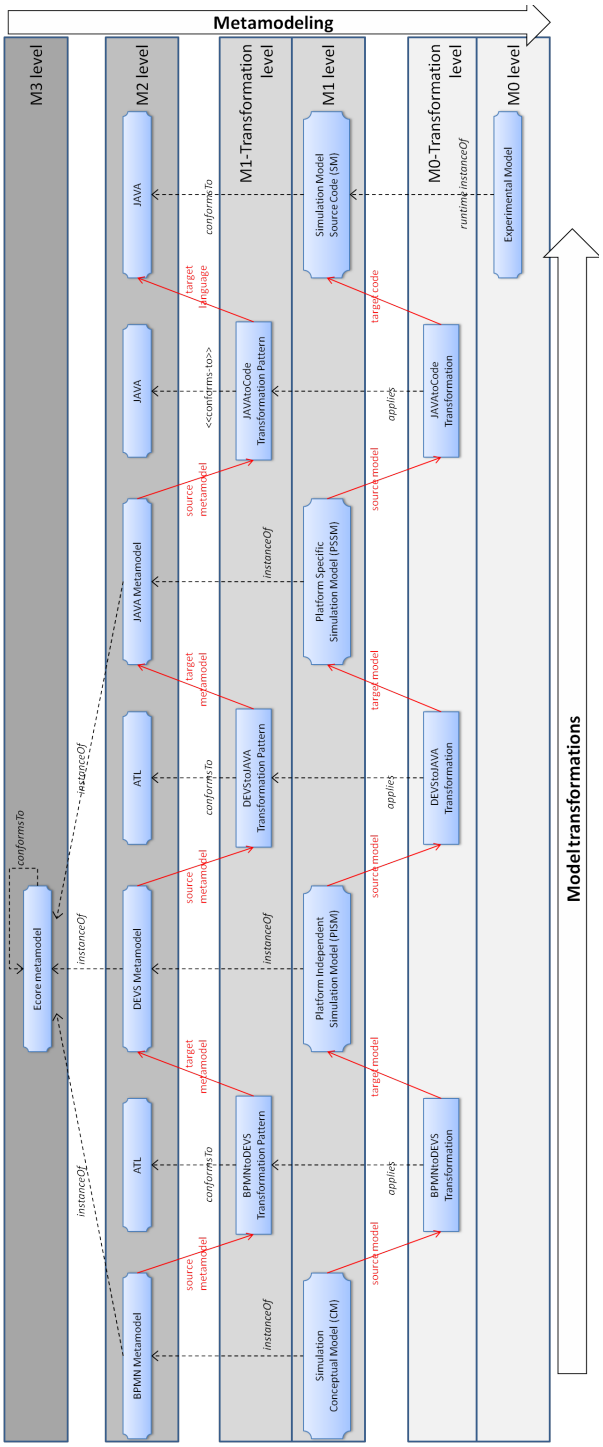


Figure 6.2: The MDD4MS process for the case study.

elements of BPMN. A visual BPMN modeling editor that works on Eclipse platform is automatically generated from this metamodel. A screenshot of the model editor is given in Figure C.12 in Appendix C.

6.5.2. DEVS metamodel

A procedural DEVS metamodel for the Hierarchical DEVS formalism with ports is defined with the GEMS plugin, and a modeling editor and an Ecore metamodel are automatically generated. The metamodel is shown in Figure 6.4. A DEVSModel represents a platform-independent simulation model. The main graphical elements of the metamodel are DEVSCoupledComp and DEVSAtomicComp, which inherit from DEVSComponent. Each DEVS component has input and output ports. Coupled models are defined hierarchically and couplings are represented via connecting the ports. Atomic models have state variables and functions as well. Functions of the atomic components are: DeltaExtFunction, DeltaIntFunction, TimeAdvanceFunction, and LambdaFunction.

The metamodel includes both structural and behavioral abstraction of the DEVS formalism. The behavior of the functions is represented via a pseudo-code metamodel that is linked by the use of an Expression attribute. Any expression can be a function call, a conditional block, or an assignment. A conditional block can be a while loop block, an if-block, or an if-else block. Each block contains other expressions. A visual DEVS modeling editor is automatically generated from this metamodel. A screenshot of the model editor is given in Figure C.13 in Appendix C.

6.5.3. JAVA metamodel

The JAVA metamodel is defined with the GEMS plugin and a modeling editor and an Ecore metamodel are automatically generated. The metamodel is shown in Figure 6.5. A JAVAmodel represents a platform-specific simulation model. The JAVAclass represents the Java classes and so each component has import definitions for the required Java packages, port definitions and a constructor. The coupled component constructor has connection definitions and subcomponent definitions. The atomic component constructor has an initialization code. In this research, the JAVA metamodel is a generic one and ignores software optimization. A detailed approach can be found in [HKGV10]. Although a visual modeling editor is automatically generated from this metamodel, it is not used for modeling purposes. The metamodel is utilized for direct code generation from a JAVA model.

6.5.4. M2M transformation from BPMN to DEVS

The BPMN-to-DEVS transformation produces atomic and coupled models with ports, couplings and templates for the system dynamics. Once the source and target metamodels are available, model transformation rules from the source models to the target models can be specified. A model-to-model transformation from BPMN to DEVS is defined by using the BPMN metamodel and DEVS metamodel.

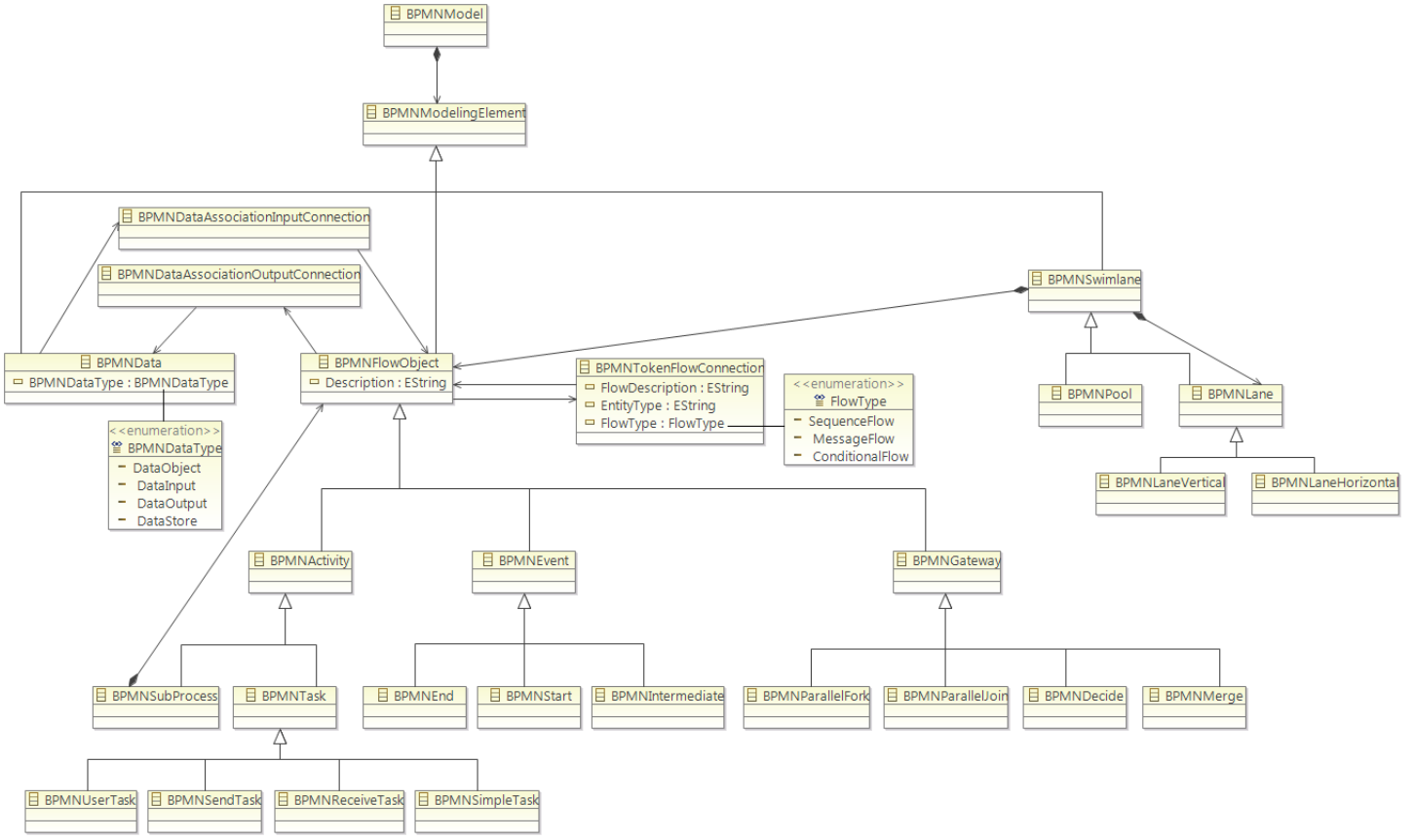


Figure 6.3: BPMN metamodel.

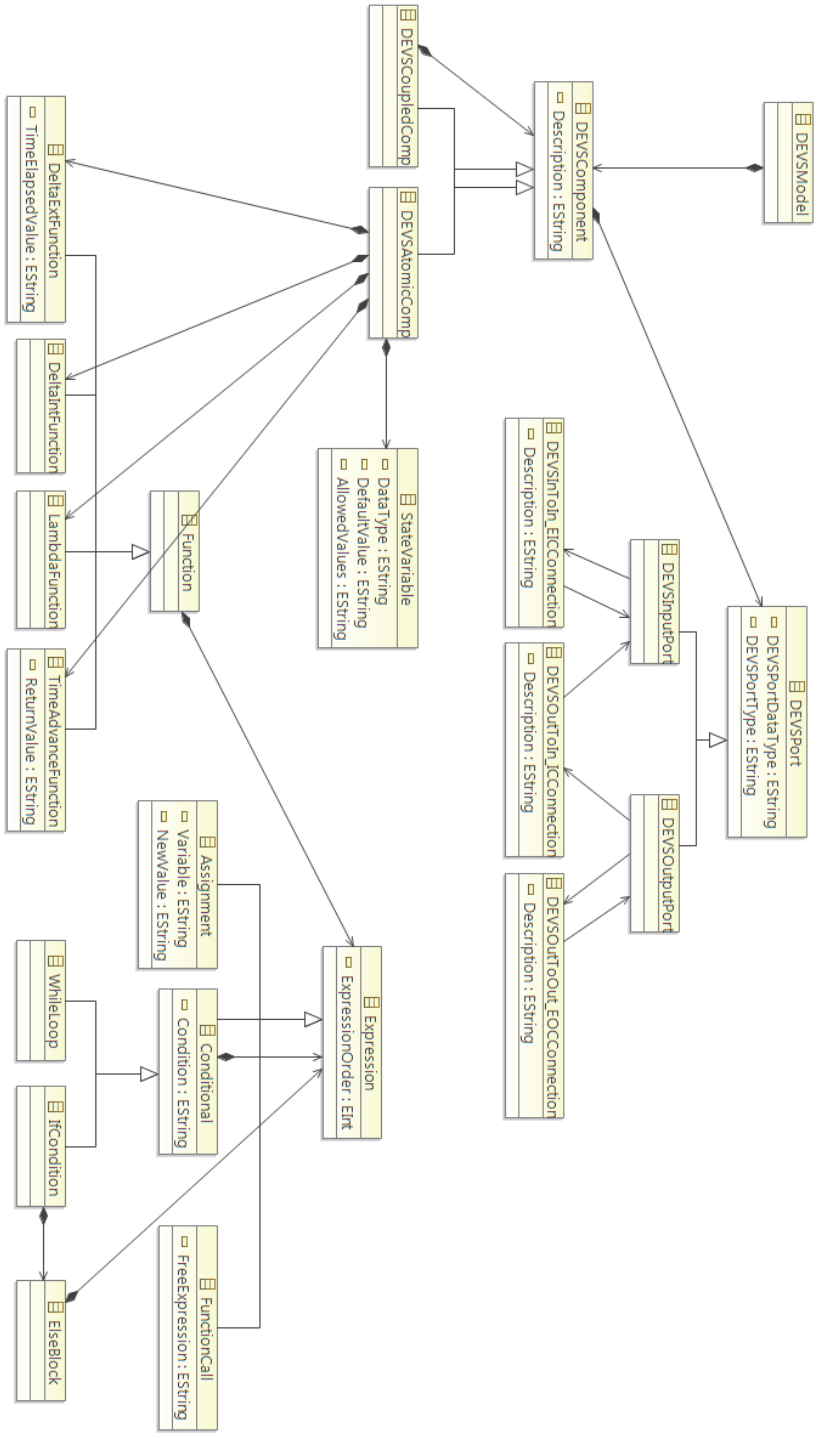


Figure 6.4: DEVS metamodel.

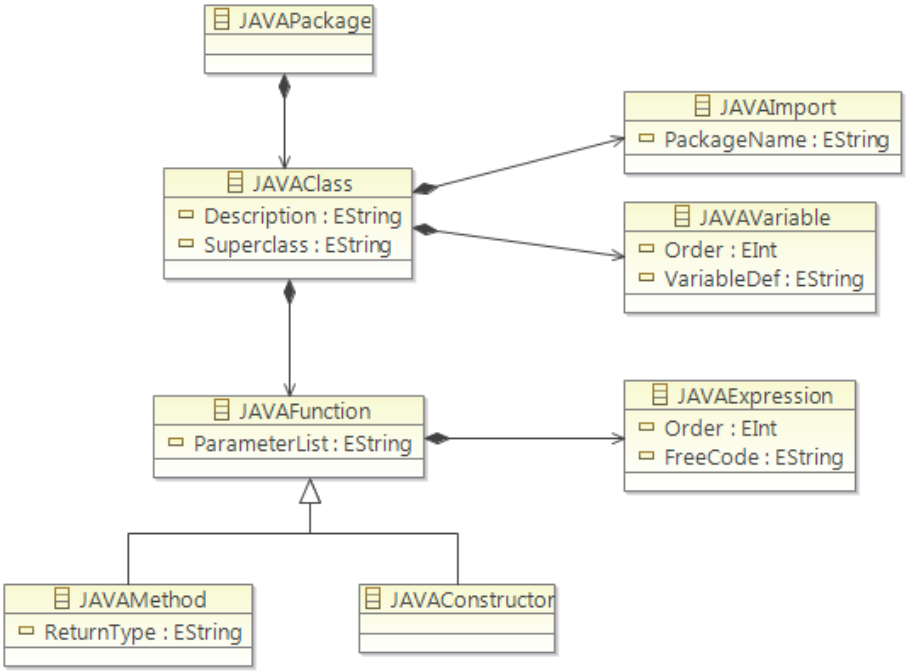


Figure 6.5: JAVA metamodel.

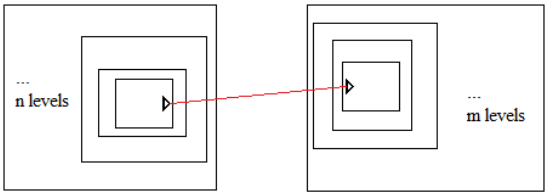


Figure 6.6: Hierarchy in a BPMN model.

The transformation is written in ATL, as proposed in the MDD4MS prototype. The transformation has two steps. In the first step, all BPMN model elements are transformed into specific DEVS model elements; and all connections are transformed into internal couplings from an output port in the source component to an input port in the target component. Ports are also generated. If the connections in the source model connect only elements of the same layer, then the output of the first step becomes a valid DEVS model and the second step is skipped. However, this does not apply in most cases. BPMN models generally have connections which cross more than one modeling elements, that is, which connect the modeling elements of different layers. Therefore, the internal couplings generated in the first step need to be refined. So, the output of the first step is only a temporary model.

In the second step, the external input couplings (EICs) and external output couplings (EOCs) are defined for the nested components. In this way, the DEVS compatibility of the target models is guaranteed. The required number of EIC and EOC is determined with the number of the nested components that a flow crosses. Figure 6.6 illustrates the all-inclusive case when the source component is nested *n* levels and the target component is nested *m* levels. In this case, *n* times DEVSOutputPort and *m* times DEVSInputPort are generated. Besides, *n* times DEVSOutToOut_EOCConnection and *m* times DEVSInToIn_EICConnection are defined. Table 6.1 shows the specific DEVS modeling elements that are generated for the BPMN modeling elements. Figure 6.7 shows a sample transformation such that a BPMN source model (a) is transformed into the temporary model (b); and then the target DEVS model (c) is generated. While BPMNSendTask, BPMNReceiveTask and BPMNSimpleTask are transformed to atomic component, BPMNUserTask is transformed into a coupled component. This is because a shared resource is needed to perform the user tasks.

Sample rules for the BPMNtoDEVS transformation are given in Listing 6.1. The source pattern starts with the keyword *from* and declares which element type of the source metamodel has to be transformed. The target pattern starts with the keyword *to* and declares into which element type(s) of the target metamodel has to be generated. It may contain one or several target pattern elements. Each target pattern element consists of a variable declaration and a sequence of bindings (assignments). These bindings consist mainly of left arrow constructs.

Table 6.1: BPMN-to-DEVS transformation pattern.

BPMN Metamodel	**DEVS Metamodel**
BPMNModel	DEVSModel
BPMNSwimlane (BPMNPool, BPMNLaneVertical, BPMNLaneHorizontal)	DEVSCoupledComp + SM_Swimlane:DEVSCoupledComp + SM_Swimlane.out: DEVSOutputPort + SM_Swimlane.outSS: DEVSOutputPort + SM_Swimlane.inS: DEVSInputPort
BPMNSendTask	DEVSAtomicComp
BPMNReceiveTask	DEVSAtomicComp
BPMNSimpleTask	DEVSAtomicComp
BPMNUserTask	DEVSCoupledComp + outServer: DEVSOutputPort + inServer: DEVSInputPort + serverStatus: DEVSInputPort
BPMNEvent (BPMNStart,BPMNEnd, BPMNIntermediate)	DEVSAtomicComp
BPMNGateway (BPMNDecide,BPMNMerge, BPMNParallelFork, BPMNParallelJoin)	DEVSAtomicComp
BPMNTokenFlowConnection	DEVSOutToIn_ICConnection + Source.out: DEVSOutputPort + Target.in: DEVSInputPort + SourceParents.EOC_Ports: DEVSOutputPort + TargetParents.EIC_Ports: DEVSInputPort

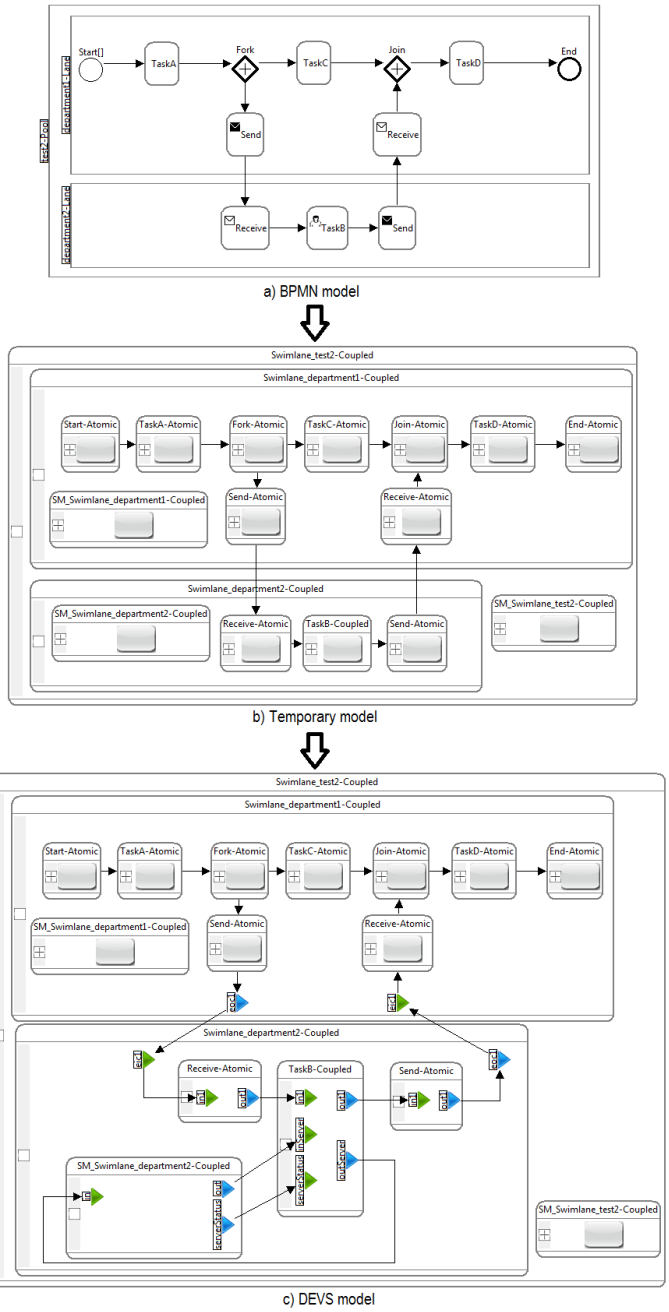


Figure 6.7: Sample model transformation from BPMN to DEVS.

Listing 6.1: Sample ATL rules for BPMNtoDEVS transformation

```
module BPMN_To_DEVS;
create OUT: SM_Metamodel from IN: CM_Metamodel;

--count parent output ports
helper context SM_Metamodel!DEVSComponent def: countOutPorts():
  Integer =
  self.DEVSPorts->select (d|d.DEVSPortType='OutputPort')->size() + 1;
--count parent input ports
helper context SM_Metamodel!DEVSComponent def: countInPorts():
  Integer =
  self.DEVSPorts->select (d|d.DEVSPortType='InputPort')->size() + 1;
...

--Transform Main Model
rule CM2SMModel {
  from
    s: CM_Metamodel!BPMNModel (true)
  to
    t: SM_Metamodel!DEVSModel (
      DEVSComponents <- s.BPMNModelingElements,
      Id <- s.Id + 1,
      Name <- s.Name
    )
}

--Transform Flow Objects
rule BPMNFlowObjectToDEVSAtomic_NOTinRoot {
  from
    s: CM_Metamodel!BPMNFlowObject
                        (not s.isInRoot() and not s.isUserTask())
  to
    t: SM_Metamodel!DEVSAtomicComp (
      DEVSParentComponent <- s.getParent(),
      Id <- s.Id + 1,
      Name <- s.Name,
      X <- s.X,
      Y <- s.Y,
      Width <- s.Width+50,
      Height <- s.Height+50,
      ExpandedWidth <- s.Width+50,
      ExpandedHeight <- s.Height+50,
      Expanded <- false,
      Description <- s.Description,
      Annotation <- s.getType()
    )
}
...
```

An attribute of the target model *t* (on the left side of the arrow) receives a return value of an Object Constraint Language (OCL) expression (on the right side of the arrow) that is based on the source model *s*. In this sense, the right side of the arrow may consist of an attribute of *s* or a call to a helper function. A helper function is an OCL expression to define global variables and functions.

6.5.5. M2M transformation from DEVS to JAVA

The DEVS-to-JAVA transformation produces valid JAVA visual models with information for Java classes. A model-to-model transformation from DEVS to JAVA is defined by using the DEVS metamodel and JAVA metamodel. The transformation is written in ATL and has two steps. In the first step, DEVSComponent instances are transformed into JAVAClass instances. Coupled component files include the package imports, class definition, port definitions, constructor definition, contained component definitions, and couplings. Coupled component files are fully transformed and they are ready for compiling. Atomic component files include imports, class definition, port definitions, and constructor definition. Also, deltaExternal(double e, Object inp), deltaInternal(), lambda(), and timeAdvance() functions are generated, which need to be refined for the user-defined expressions.

The generated JAVAclass extends from either AtomicModel or CoupledModel abstract classes in the DEVSDSOL library. The generated JAVA models include all the required information to generate source code. In the second step of the transformation, the parent function of each expression is redefined to clearly link the model parts. Table 6.2 shows the specific JAVA modeling elements that are generated for the DEVS modeling elements. Figure 6.8 shows a sample transformation for an atomic DEVS model. A sample rule from the ATL transformation is given in Listing 6.2.

Listing 6.2: Sample ATL rule for DEVStoJAVA transformation

```
rule DEVSCoupled2JavaClass {
  from
    s: DEVS_Metamodel!DEVSCoupledComp (true)
  to
    t: JAVA_Metamodel!JAVAClass (
      Id <- s.Id + 1,
      Name <- s.Name,
      X <- s.X,
      Y <- s.Y,
      ...
      Description <- s.Description,
      Superclass <- 'CoupledModel',
      JAVAVariables <- s.DEVSPorts
    ),
    subComp: JAVA_Metamodel!JAVAExpression (
      Name <- s.Name,
      FreeCode <- s.Annotation + ' var_' + s.Name + ' = new ' +
              s.Annotation + '(this, "' + s.Name + '", "' +
              s.Description + '", ' + thisModule.countID + ');',
      JAVAParentFunction <- s.getParent()
    ),
    c: JAVA_Metamodel!JAVAConstructor (
      Id <- s.Id + 2,
      Name <- s.Name,
      ParameterList <- 'CoupledModel parentModel, String name,
                                    String desc, Integer id'
    ),
    exp: JAVA_Metamodel!JAVAExpression (
      Name <- 'superCall',
      FreeCode <- 'super(name, parentModel);\n',
      JAVAParentFunction <- c
    )
  do {
    thisModule.DefineConstructor(t, c);
    thisModule.incCountID();
    thisModule.AssignFunction(c, exp);
  }
}
```

6.5.6. Code generation from the JAVA model

In the last step, a code generator for JAVA models is used to generate the source code automatically. The code generator is a visitor-based model interpreter and has been written in Java. Java files for each DEVS component are generated separately. A part of the model interpreter code is given in Listing 6.3. The visitClass function calls the visitVariables and visitMethods functions.

Table 6.2: DEVS to JAVA transformation pattern.

DEVS Metamodel	**JAVA Metamodel**
DEVSModel	JAVAPackage + TestModel: JAVAClass + TestModel: JAVAConstructor + JAVAExpressions
DEVSCoupledComp	JAVAClass + JAVAConstructor + JAVAExpressions
DEVSAtomicComp	JAVAClass + JAVAConstructor + JAVAExpressions
DEVSInputPort	JAVAVariable
DEVSOutputPort	JAVAVariable
StateVariable	JAVAVariable
DEVSOutToIn_ICConnection	JAVAExpression
DEVSInToIn_EICConnection	JAVAExpression
DEVSOutToOut_EOCConnection	JAVAExpression
Expression	JAVAExpression
DeltaIntFunction	JAVAMethod
DeltaExtFunction	JAVAMethod
LambdaFunction	JAVAMethod
TimeAdvanceFunction	JAVAMethod

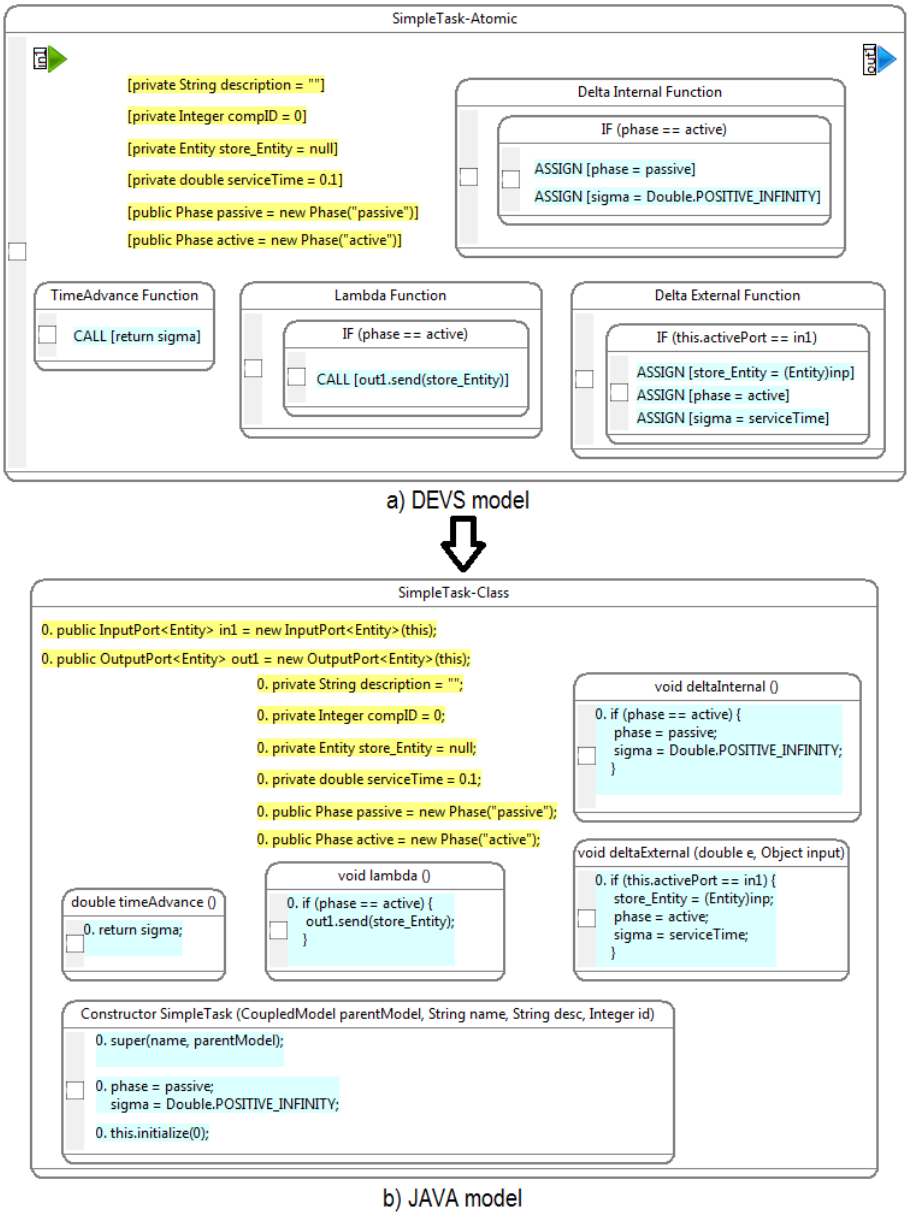


Figure 6.8: Sample model transformation from DEVS to JAVA model.

Listing 6.3: Sample code from the JAVA model interpreter

```
public class JAVA_to_JAVACode_Interpreter
      extends org.eclipse.gmt.gems.model.actions.AbstractInterpreter
      implements javacode.javamodeling.JAVAModelingVisitor {

  private static final long serialVersionUID = 1L;
  String projectPath = "../";
  FileWriter outCurrent = null;

  //constructor
  public JAVA_to_JAVACode_Interpreter() {
    super("javacode.javadiagram.JAVA_to_JAVACode_Interpreter");
  }

  @Override
  public void visitJAVAClass(JAVAClass tovisit) {
    System.out.println("Visiting a DEVS Model");
    String CCname = tovisit.getName();
    String filePath = projectPath + CCname + ".java";
    File a = new File(filePath);

    try {
      outCurrent = new FileWriter(a);
    } catch (IOException e1) {
      e1.printStackTrace();
    }

    //start writing
    write_file("package bpmnmodel;\n", 0);

    //get imports
    write_file("import bpmnlibrary.*;", 0);
    write_file("import queueLibrary.*;", 0);
    write_file("import nl.tudelft.simulation.dsol.formalisms.devs.
                                    ESDEVS.CoupledModel;", 0);
    write_file("import nl.tudelft.simulation.dsol.formalisms.devs.
                                    ESDEVS.InputPort;", 0);
    write_file("import nl.tudelft.simulation.dsol.formalisms.devs.
                                    ESDEVS.OutputPort;\n\n", 0);
    write_file("/**", 0);
    write_file(" * " + CCname + " class", 0);
    write_file(" * ", 0);
    write_file(" * " + tovisit.getDescription() +" <br><br>" , 0);
    write_file(" * @version 1.0 <br>", 0);
    write_file(" * ", 0);
    write_file(" * @author Auto-generated with JAVA Code generator
                                                    <br>", 0);
    write_file(" */", 0);
```

```
  write_file("public class " + CCname + " extends " +
                          tovisit.getSuperclass() + " {\n", 0);

  //write variables
  write_file("private static final long serialVersionUID
                                                  = 1L;\n", 1);
  visitVariables(tovisit);
  write_file("\n", 0);

  //constructor
  write_file("/** ", 1);
  write_file(" * Constructor for " + CCname, 1);
  write_file(" * ", 1);
  write_file(" */ ", 1);
  visitJAVAConstructor(this.getConstructor(tovisit));

  //write other functions and finish
  visitMethods(tovisit);
  write_file("}", 0);

  try {
    outCurrent.close();
  } catch (IOException e1) {
    e1.printStackTrace();
  }
  System.out.println("---> Finished a JAVA Class.");
}

@Override
public void visitJAVAConstructor(JAVAConstructor tovisit) {
  write_file("public " + tovisit.getName() + " (" +
                      tovisit.getParameterList() + ") {", 1);
  visitJAVAExpressions (tovisit);
  write_file("}", 1);
}

@Override
public void visitJAVAExpression(JAVAExpression tovisit) {
  write_file(tovisit.getFreeCode(), 2);
}

@Override
public void visitJAVAMethod(JAVAMethod tovisit) {
  write_file("@Override", 1);
  write_file("protected " + tovisit.getReturnType() + " " +
                      tovisit.getName() + " (" +
                      tovisit.getParameterList() + ") {" , 1);
  write_file("// TODO Auto-generated block", 2);
```

```
    visitJAVAExpressions (tovisit);
    write_file("}\n", 1);
  }

  @Override
  public void visitJAVAVariable(JAVAVariable tovisit) {
    write_file(tovisit.getVariableDef(), 1);
  }
}
```

The code generator is added as an extension to the JAVA modeling editor and it can be called for each model from a right click menu. After the code generation, the generated code can be compiled. If there are manually entered code pieces they need to be checked for compilation errors such as try/catch blocks or missing imports. After fixing these kind of compile errors, the simulation model is ready to be run. Listing 6.4 shows the generated code for the example in Figure 6.8.

Listing 6.4: Generated code for the model in Figure 6.8.

```
package TestModel;

import nl.tudelft.simulation.dsol.formalisms.devs.ESDEVS.AtomicModel;
import nl.tudelft.simulation.dsol.formalisms.devs.ESDEVS.CoupledModel;
import nl.tudelft.simulation.dsol.formalisms.devs.ESDEVS.InputPort;
import nl.tudelft.simulation.dsol.formalisms.devs.ESDEVS.OutputPort;
import nl.tudelft.simulation.dsol.formalisms.devs.ESDEVS.Phase;

/**
 * SimpleTask class
 * Atomic Model that implements BPMN Simple Task <br><br>
 * @version 1.0 <br>
 * @author Auto-generated with JAVA Code generator <br>
 */
public class SimpleTask extends AtomicModel {

  private static final long serialVersionUID = 1L;

  public InputPort<Entity> in1 = new InputPort<Entity>(this);
  public OutputPort<Entity> out1 = new OutputPort<Entity>(this);
  private String description = "";
  private Integer compID = 0;
  private Entity store_Entity = null;
  private double serviceTime = 0.1;
  public Phase passive = new Phase("passive");
  public Phase active = new Phase("active");

  /**
   * Constructor for SimpleTask
   */
```

```
  public SimpleTask (CoupledModel parentModel, String name,
                                        String desc, Integer id) {
    super(name, parentModel);
    phase = passive;
    sigma = Double.POSITIVE_INFINITY;
    this.initialize(0);
  }

  @Override
  protected void deltaInternal () {
    // TODO Auto-generated block
    if (phase == active) {
     phase = passive;
     sigma = Double.POSITIVE_INFINITY;
    }
  }

  @Override
  protected void deltaExternal (double e, Object input) {
    // TODO Auto-generated block
    if (this.activePort == in1) {
     store_Entity = (Entity)inp;
     phase = active;
     sigma = serviceTime;
    }
  }

  @Override
  protected double timeAdvance () {
    // TODO Auto-generated block
    return sigma;
  }

  @Override
  protected void lambda () {
    // TODO Auto-generated block
    if (phase == active) {
     out1.send(store_Entity);
    }
  }
}
```

6.5.7. Using a DEVS simulation model component library for BPMN

In order to support the transformation process and to generate fully executable DEVS models, a DEVS simulation model component library for BPMN is proposed in [RÇSW11]. The library is improved throughout this research. Each BPMN modeling element in the BPMN metamodel has been implemented as a DEVS

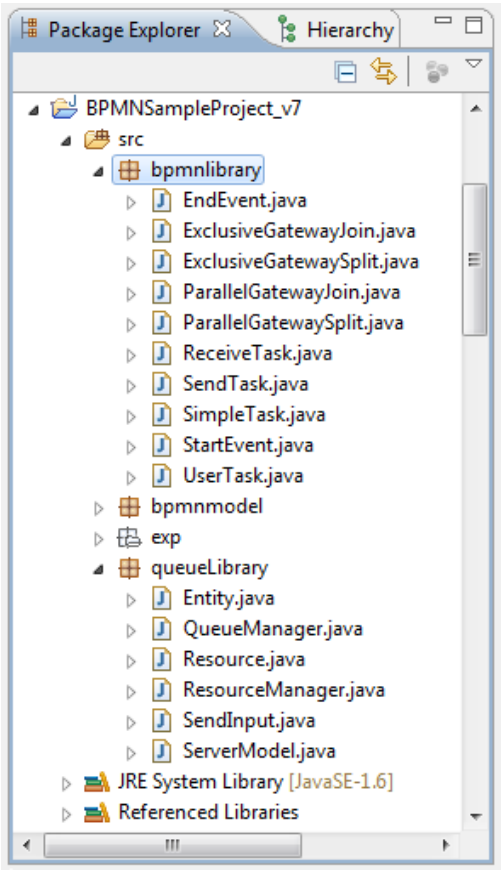


Figure 6.9: The DEVS simulation model component library for BPMN.

component in Java and these elements are executable with DEVSDSOL simulation library. The components satisfy the requirements given in Section 5.2.2 and the components' behavior are validated. Usability evaluation of the MDD4MS prototype and the DEVS component library with business process modelers is presented in [Rus11]. A general overview of the library is shown in Figure 6.9. More details about each component are given in Appendix C.

The library includes a queuing library and a resource allocation mechanism as well. Each component has been linked to a PISM template. Hence, the transformation process can fully be automated. This means that a BPMN model can be successfully transformed into an executable DEVS simulation model which is written in JAVA, and so it can be executed via automated model transformations.

6.5.8. Model 1: The customer service process of a telecom operator

The MDD4MS framework and the prototype has been successfully tested and used in a case study at Accenture Netherlands [Rus11]. Accenture is a global management consulting, technology services and outsourcing company serving in more than 120 countries [Acc13]. The role of Accenture management consulting activities is generally speaking to advise and support clients with their business decisions and support possible business transformations. One of the projects that Accenture undertook was for a telecom operator to support decisions with regard to the roll out of fiber optic cables in the Netherlands of over 100 administrative areas. The purpose of the project was to help the organization develop robust operational and tactical business plans and continuously improve the speed and quality of analysis and decision making. In this section, a simplified version of the Accenture's customer service process model is presented. The MDD4MS prototype has been used to develop the simulation model according to the MDD4MS framework.

Problem definition

We will develop a discrete event simulation model for the customer service process of new orders. We will measure the average lead time and waiting time for an order to analyze the system. Orders can be either for repair or a new installation. There are three types of participants in the system which are customer service office, service technician and supply technician. The customer service is accepting the orders from the customers and sending them to the related technician. Then, a technician processes the order and schedules a date with the customer. On the scheduled date, if the customer is at home, the technician does either the repair order or the install order. For the installation process, it is straightforward and supply technician finishes his/her work. For the repair process, there is a chance that the service technician cannot complete the order and so he/she reports the problem to customer service. If both technician completes the order, then they send a billing information to the customer office. More information can be found in [Rus11].

Conceptual modeling

A conceptual model for this example is developed according to the static model given in [Rus11]. We used the BPMN editor of the MDD4MS prototype to specify the conceptual model with BPMN. Figure 6.10 shows the BPMN model.

Participants are modeled with BPMNSwimlanes. Sending activities are modeled as BPMNSendTask and receiving activities are modeled as BPMNReceiveTask. If an activity is performed by a queued set of resources then it is modeled as BPMNUserTask. Other activities are modeled as BPMNSimpleTask. Exclusive fork is used to control the flow of the application. The next section explains how the BPMN model is used in the next stages.

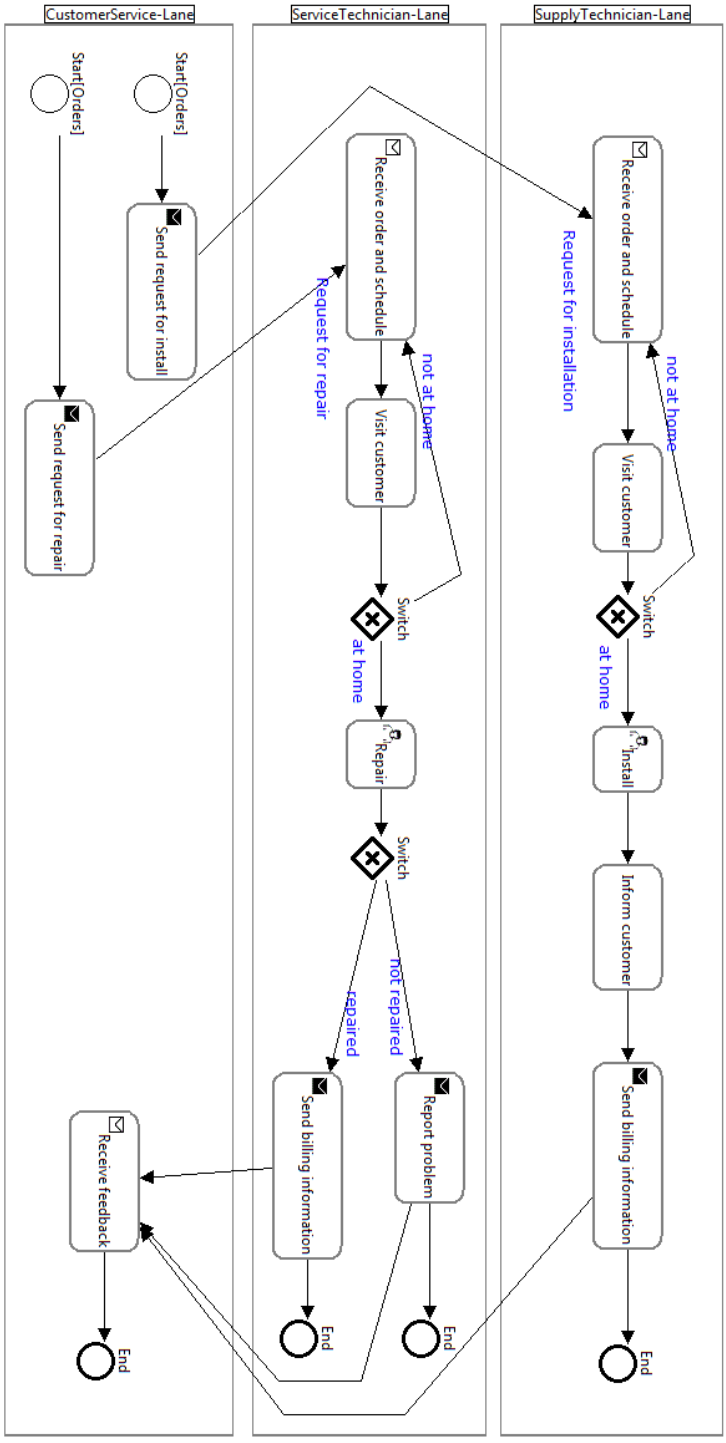


Figure 6.10: BPMN model for the customer service process.

Simulation model specification

After the BPMN model is specified, it is transformed to a DEVS model by using the BPMNtoDEVS transformation in the MDD4MS prototype. During the transformation, BPMNSwimlanes are transformed to coupled models each containing a set of resource and a resource manager by default. The activities except BPMNUserTask are transformed to atomic models which are linked to the pre-developed DEVS components in a library. The DEVS metamodel in the MDD4MS prototype is a procedural metamodel and it includes a simple pseudo code mechanism. So, it is possible to generate the transition functions of an atomic component with the model transformation rules. However, in this case, we preferred to use a pre-developed DEVS library for BPMN which is provided with the MDD4MS prototype. A BPMNUserTask is transformed to a coupled model which contains a queue manager. All of the internal and external couplings are generated. Also, the couplings between a queue manager in a BPMNUserTask and a resource manager in its parent swimlane are defined. The auto-generated DEVS model is an instance of the DEVS metamodel and it can be viewed with the DEVS editor in the MDD4MS prototype. Figure 6.11 shows the DEVS model. At this point, the model can be changed or improved if needed.

Model implementation

After the DEVS model is generated, it is transformed to a Java model by using the DEVStoJAVA transformation in the MDD4MS prototype. During the transformation, all components are transformed to Java classes. All input ports, output ports and state variables are transformed to Java variables. All expressions in the pseudo-code mechanism are transformed to Java expressions. Constructors, methods, and parameters are defined as well. The auto-generated Java model is an instance of the JAVA metamodel. Although it can be viewed and edited with the Java model editor in the MDD4MS prototype, its main purpose is to generate the Java code. Figure 6.12 shows a screen shot from the Java model editor. By right clicking on the model within the editor and choosing the Java Code Generator menu item, the Java code is generated into a chosen folder.

Figure 6.13 shows the full Java code generated for the customer service coupled model. Since the parameters have not been specified yet, the source code has error messages. In some cases, the components have default parameters. For example, for simple task component we use default service time. However, for start event, the modeler needs to define the arrival rate.

Simulation

We use the DEVSDSOL simulation library and the pre-developed DEVS library for BPMN to execute the generated Java code, i.e. to simulate the final executable simulation model. In order to execute the model, we define the experimental model

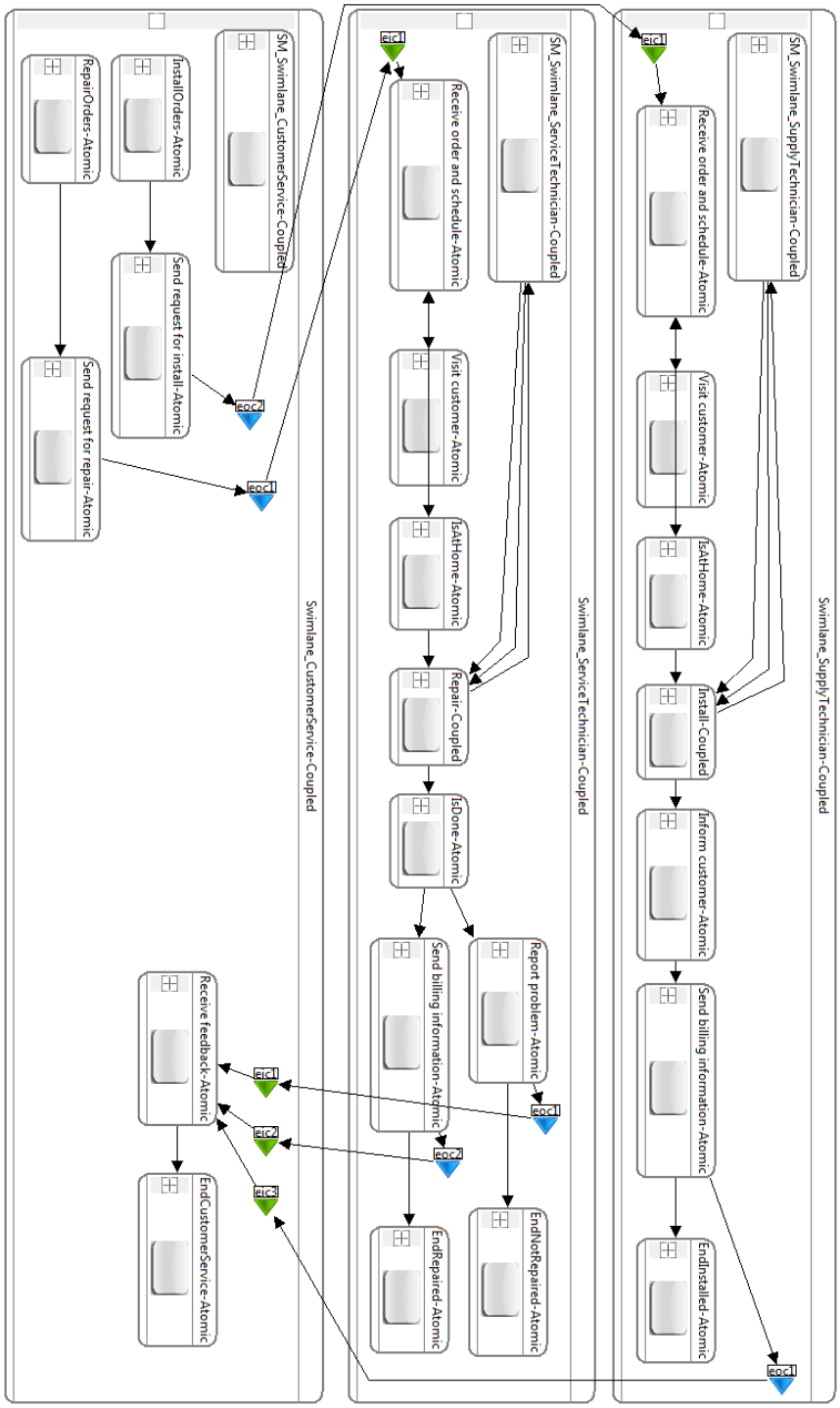


Figure 6.11: Auto-generated DEVS model for the customer service process.

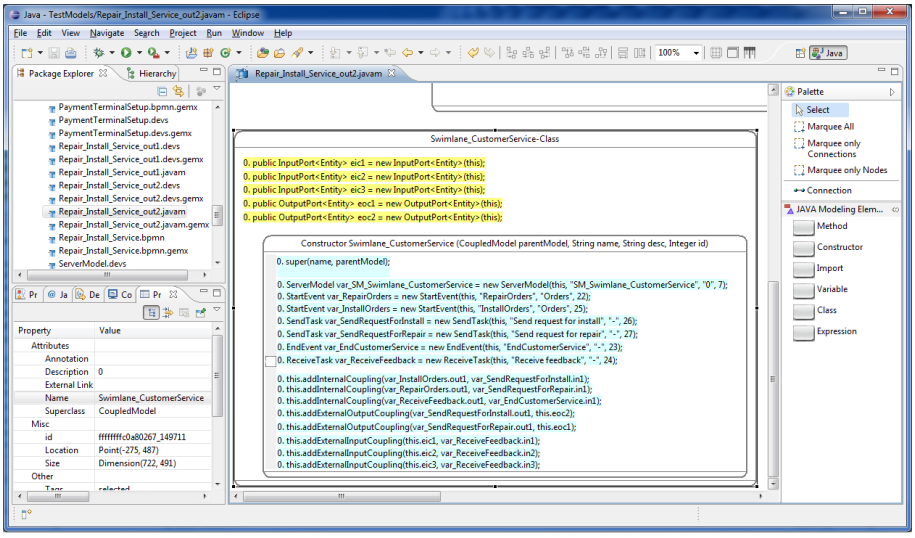


Figure 6.12: Auto-generated JAVA visual model for a coupled model.

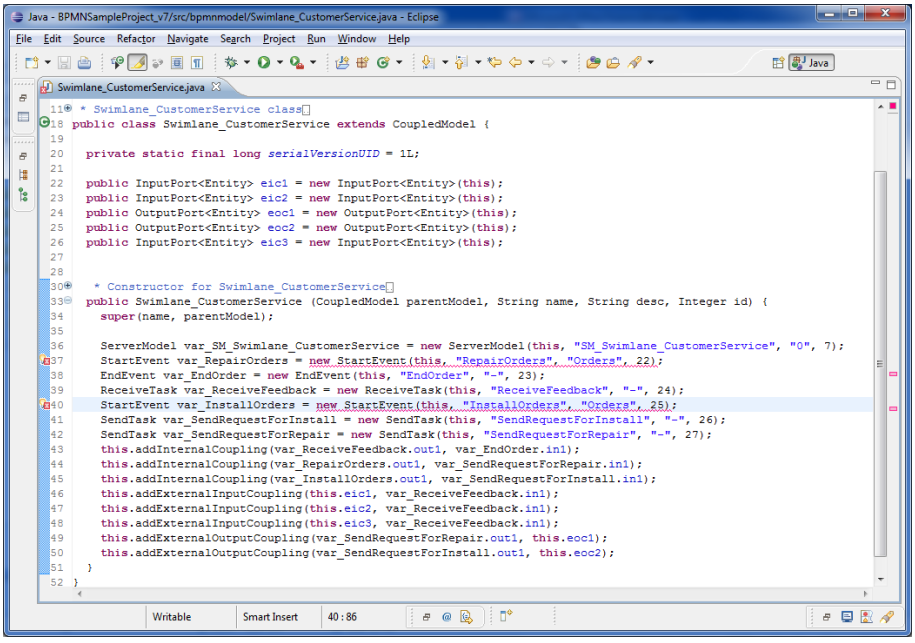


Figure 6.13: Auto-generated JAVA source code for a coupled model.

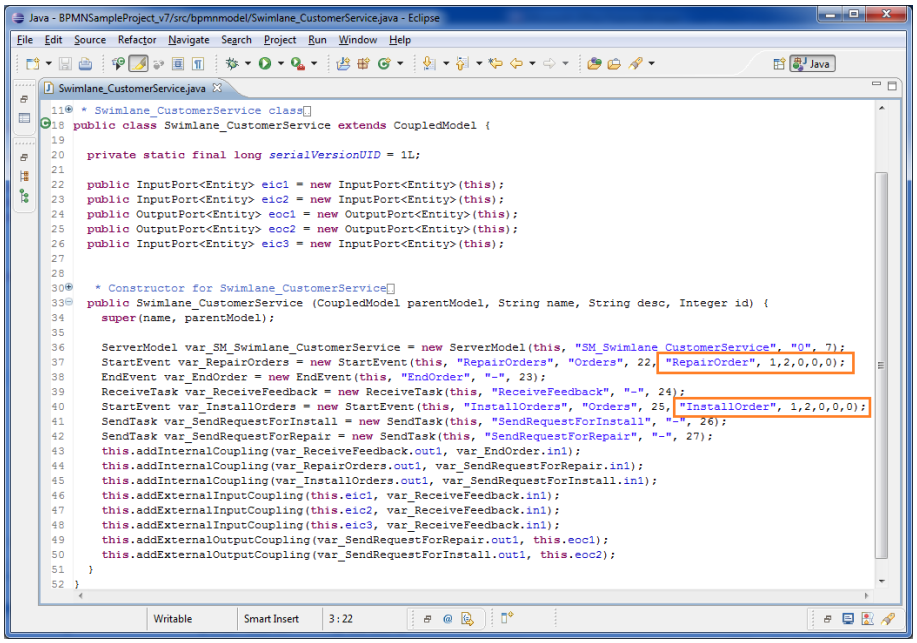


Figure 6.14: Adding the necessary parameters for the experimental model.

by adding the required parameters such as the arrival rate, queue capacity, service rates, etc. in the Java code. We add the parameters in the source code as shown in Figure 6.14. Appendix C provides more information about the parameter list of the component constructors. Although, it is possible to set the parameters via a user interface, in this project we prefer to add them manually due to time and cost limitations of the research. Once we define all of the parameters and run length, we execute the model. Appendix D shows the experimental parameters and setup values. Figure 6.15 shows a screen shot of the simulation run with the DEVS library for BPMN. For this example, we measure the average lead time and waiting time for an order.

6.5.9. Model 2: The application process to obtain a working payment terminal

We have chosen a larger example from the electronic payments sector which is presented in [SBV+09]. This case study provides a good example of modeling and simulation in a complex multi-actor environment with technological interdependencies. The crucial role of modeling within this example is to document the business processes as much as possible in a visualized way, to enable different parties to gain insight into the issues and the potential solutions. We have used the MDD4MS prototype to develop the simulation model according to the MDD4MS framework.

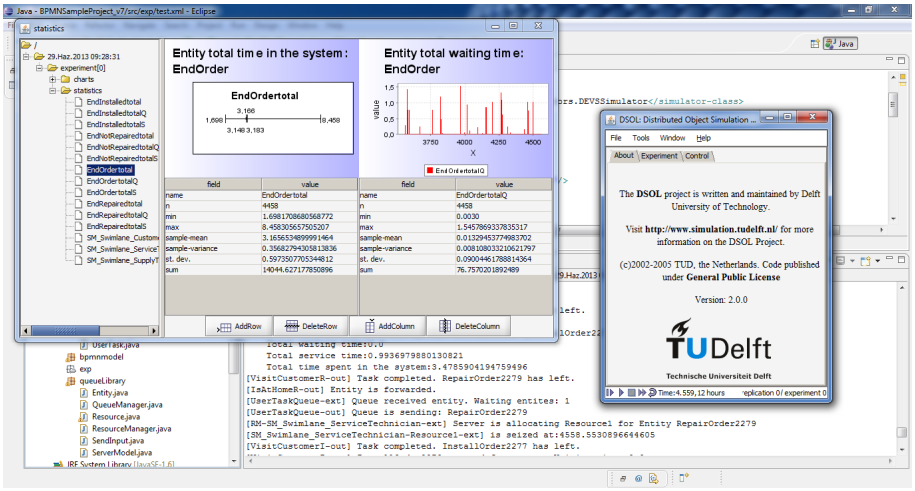


Figure 6.15: Running the auto-generated code with the DEVS library for BPMN.

Problem definition

We will develop a discrete event simulation model for the application process of new merchants to obtain a working payment terminal to accept electronic payments. There are five types of participants in the system and there are relations and dependencies between them for the major part of the system. The participants are: merchant, terminal supplier, telecom supplier, acquirer and acquiring processor. The merchant is crucial for making the application possible. The merchant sends applications to the terminal supplier, the acquirer and the telecom supplier. Then, the applications are processed by the resources of the related participants. Besides, the application information is entered to the acquiring processor database and terminal management system. When the terminal and the database of the acquiring processor contain the same information, the terminal can start accepting electronic payments. Details of the example can be found in [SBV+09].

Conceptual modeling

A conceptual model for this example is developed according to the static model given in [SBV+09]. We used the BPMN editor of the MDD4MS prototype to specify the conceptual model with BPMN. Similar to the first model, participants are modeled with BPMNSwimlanes. Sending activities are modeled as BPMNSendTask and receiving activities are modeled as BPMNReceiveTask, and so on. Parallel forks and joins are used to control the flow of the application. Figure 6.16 shows the BPMN model. A part of the BPMN model, namely the telecom supplier swimlane, is shown in Figure 6.17-a to illustrate the model continuity. The next section

explains how the BPMN model is used in the next stages.

Simulation model specification

Similar to the first model, after the BPMN model is specified, it is transformed to a DEVS model by using the BPMNtoDEVS transformation in the MDD4MS prototype. Figure 6.18 shows the DEVS model. As an example, Figure 6.17-b shows the auto generated coupled model for the telecom supplier swimlane in the BPMN model.

Model implementation

After the DEVS model is generated, it is transformed to a Java model by using the DEVStoJAVA transformation in the MDD4MS prototype. Figure 6.17-c shows the Java class model for the telecom supplier as an example. Figure 6.19 shows a screenshot from the Java model editor. Figure 6.20 shows the full Java code generated for the telecom supplier coupled model. Figure 6.17 illustrates how modeling relation is preserved during the model transformations and model continuity is obtained.

Simulation

In order to execute the model, we define the experimental model by adding the required parameters. Appendix D shows the experimental parameters and setup values. We defined arrival rate for new merchants and service rates for user tasks. Figure 6.21 shows a screen shot of the simulation run with the DEVS library for BPMN. The simulation results show how the information in the conceptual model is preserved and moved into the simulation model. For example, in the output window, '[SendConfirmation-out]' message is automatically generated and it includes the task name defined at the conceptual modeling stage.

6.6. Evaluation of the case study

In this section, we will examine the generated models and results at the case study. We would like to analyze how conceptual modeling stage is effected by the application of the MDD4MS framework, and if model continuity between the different models of the M&S lifecycle is obtained.

6.6.1. MDD4MS checklist

In this section, we will evaluate the case study according to the MDD4MS checklist given in Section 3.5. The following tables show the summary of the presented case study for applying MDD4MS in practice.

Table 6.3 shows the information about the languages and the metamodels in the case study. Table 6.4 shows the information about the tools in the case study.

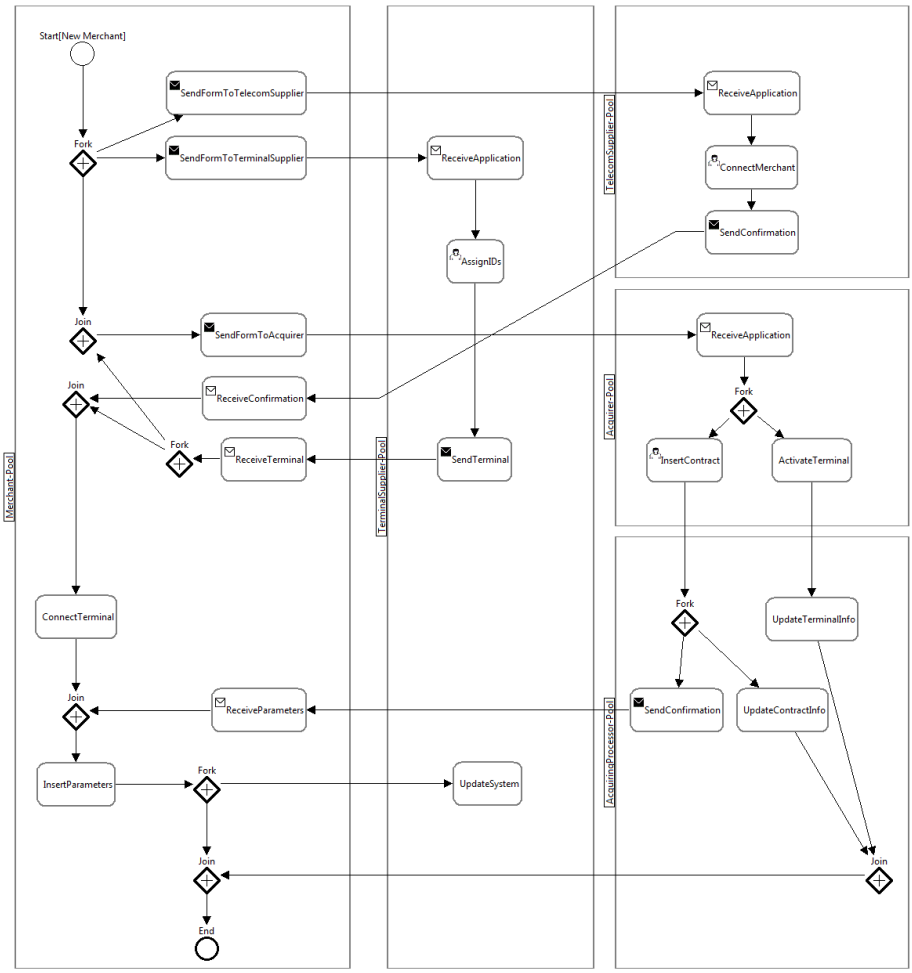
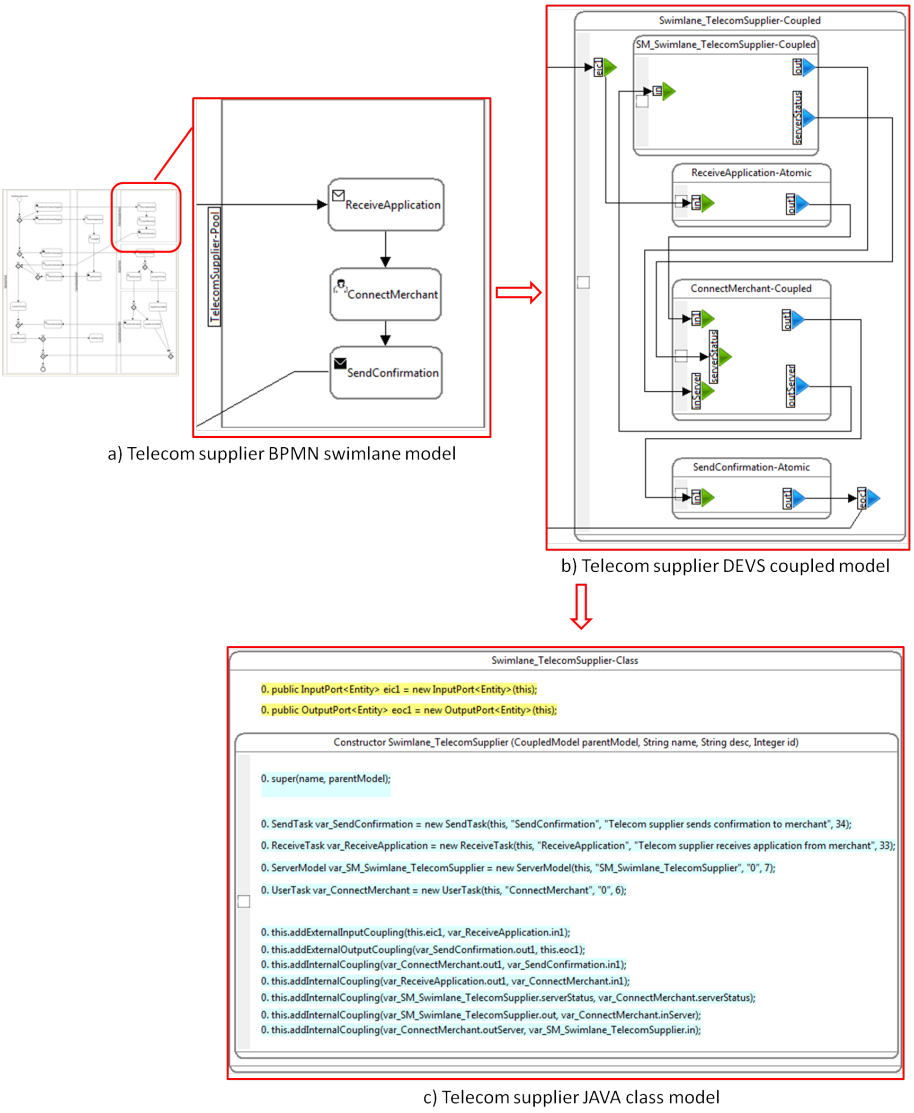


Figure 6.16: BPMN model for the terminal application process.



a) Telecom supplier BPMN swimlane model

b) Telecom supplier DEVS coupled model

c) Telecom supplier JAVA class model

Figure 6.17: Model continuity in the case example.

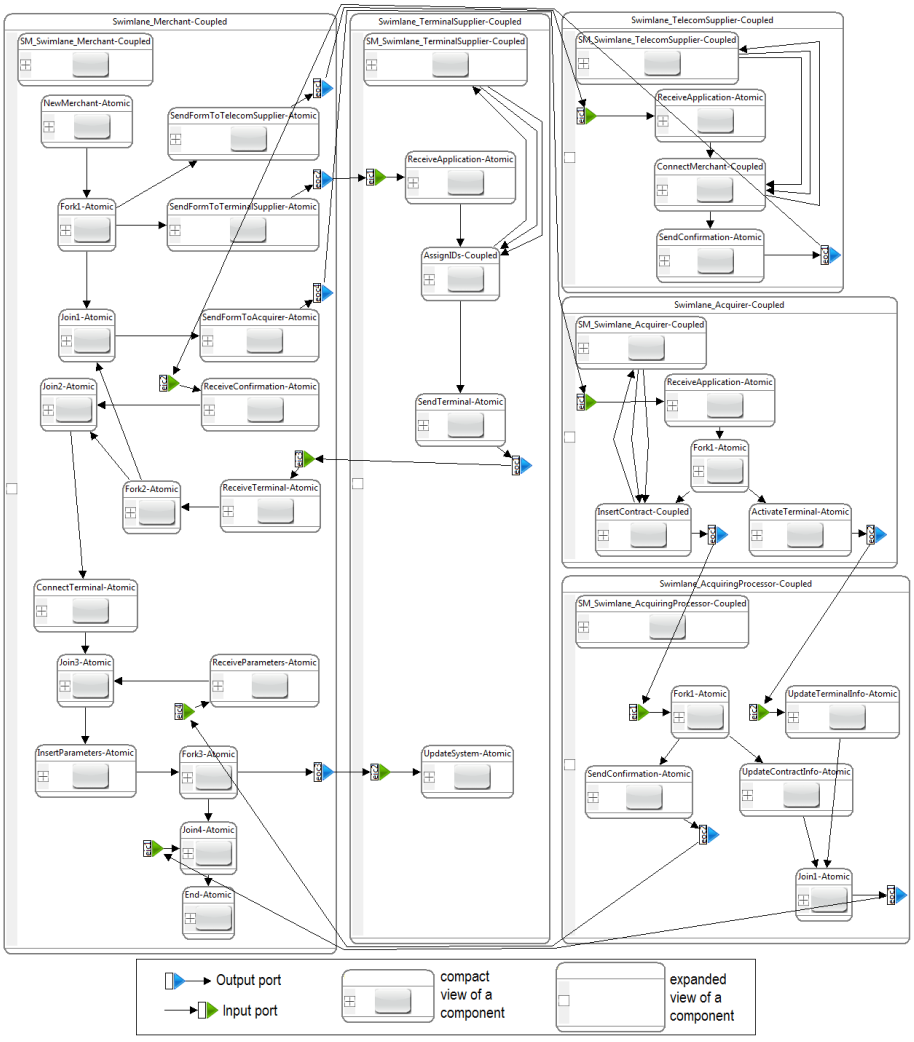


Figure 6.18: Auto-generated DEVS model for the terminal application process.

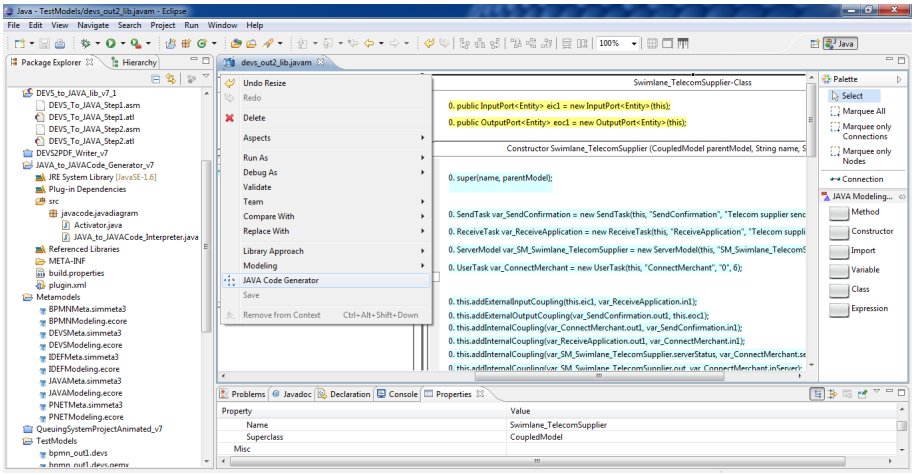


Figure 6.19: Auto-generated JAVA visual model for a coupled model.

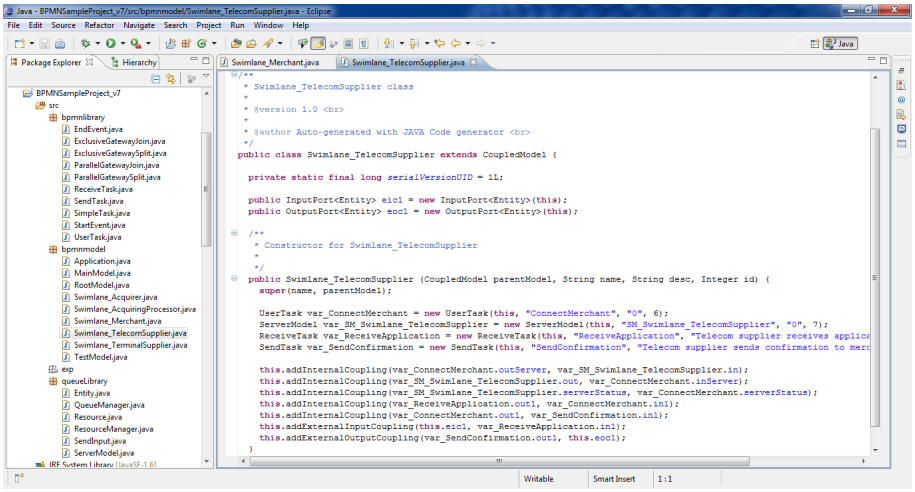


Figure 6.20: Auto-generated JAVA source code for a coupled model.

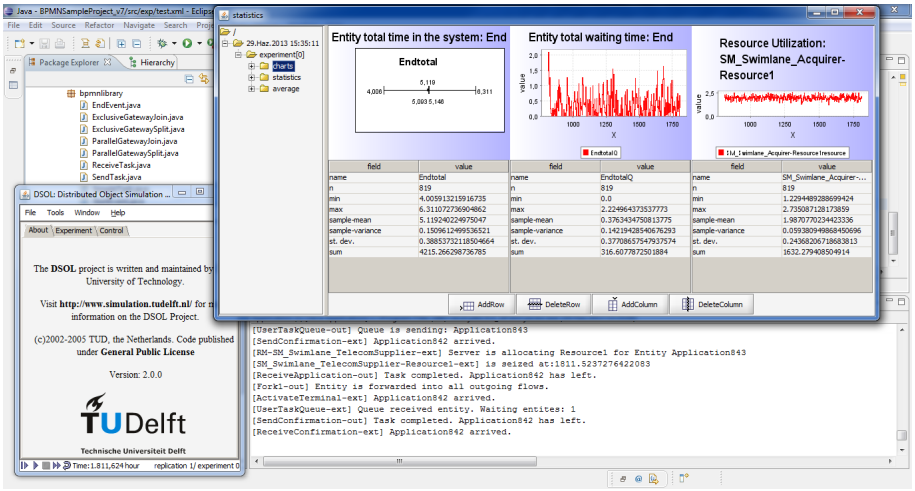


Figure 6.21: Running the auto-generated code with the DEVS library for BPMN.

Table 6.5 shows the information about the developed or generated models, transformations and other artifacts in the case study. The tables show that we covered all of the steps in the framework.

6.6.2. Validation of the results

We use the simulation software Arena to validate the outcomes of the auto generated simulation models. Arena is a widely used simulation software and it is considered to provide correct results [Are13]. The choice for Arena was made because it is possible to transform the business process modeling concepts easily to Arena modules. For example, a start event can be modeled with a Create module, and end event can be modeled with a Dispose module. A task can be modeled with a Process module, and so on. During the validation, for a given BPMN model, a DEVS model was automatically generated as well as an Arena simulation model was developed manually. The Arena models are developed independently and validated in different research studies [SBV+09, RÇSW11]. Both the DEVS simulation models and the Arena simulation models were executed by using the same experimental data.

In this section, we compare the simulation results statistically by applying an independent two-sample t-test. This test allows us to compare the means of the two data sets. We have to take into account that the generation of random values is most likely different in DSOL and Arena, due to for instance the implemented pseudo random number generators and seed values. However, when we run the model for longer run lengths and have more replications, the means will approach each other.

Table 6.3: Applying the MDD4MS framework for discrete event simulation of business process models: languages and metamodels.

Activities	**Done? Y/N**	**artifact/ chosen method**
Choose the conceptual modeling language	Y	BPMN
Choose the system specification formalism	Y	DEVS
Choose the simulation programming language	Y	Java
Choose a metamodeling language	Y	Ecore
Define/choose the simulation conceptual modeling metamodel (CMmetamodel)	Y	BPMN metamodel
Define/choose the simulation model specification metamodel (PISMmetamodel)	Y	DEVS metamodel
Define/choose the simulation model implementation metamodel (PSSMmetamodel)	Y	JAVA metamodel
Choose a M2M transformation language	Y	ATL
Choose a M2T transformation language	Y	Java

Table 6.4: Applying the MDD4MS framework for discrete event simulation of business process models: tools.

Activities	**Done? Y/N**	**artifact/ chosen method**	**Notes**
Choose a metamodeling environment	Y	Eclipse GEMS plugin	
Choose a M2M transformation tool	Y	Eclipse ATL IDE	
Choose a M2T transformation tool	Y	Eclipse	Java program
Generate/choose the simulation conceptual model editor	Y	BPMN editor	auto generated
Generate/choose the simulation model specification editor	Y	DEVS editor	auto generated
Generate/choose the simulation model implementation editor	Y	JAVA model editor	auto generated
Choose a simulation platform	Y	Eclipse	by using DSOL and DEVSDSOL libraries

Table 6.5: Applying the MDD4MS framework for discrete event simulation of business process models: models, transformations and results.

Activities	**Done? Y/N**	**artifact/ chosen method**
Define/choose the CM-to-PISM transformation	Y	bpmn2devs.atl
Define/choose the PISM-to-PSSM transformation	Y	devs2java.atl
Define/choose the PSSM-to-Code transformation	Y	java2code.java
Specify the CM	Y Y	RepairInstallService.bpmn PaymentTerminalSetup.bpmn
Generate and refine the PISM	Y Y	RepairInstallService.devs PaymentTerminalSetup.devs
Generate and refine the PSSM	Y Y	RepairInstallService.javam PaymentTerminalSetup.javam
Generate and refine the SM	Y	auto generated Java classes
Design experiments	Y	Experimental models
Execute the SM	Y	Simulation results

Model-1, i.e. the customer service process model, was executed for 30 replications. Each replication runs for 5000 hours with a 100 hours warm-up period. Model-2, i.e. the payment terminal application process model, was executed for 25 replications. Each replication runs for 2000 hours with a 50 hours warm-up period. Appendix D presents the experimental parameters and the results for the models.

With the t-test, we expect to show that the sample data sets are similar and there is no statistically significant difference between them. Hence, our null hypothesis for t-test is that the results for the two simulation models are different. We use the SPSS statistical data analysis software [IBM13]. During the t-test, we focus on the average total time in the system, average waiting time in user task queues and the resource utilization statistics. After having the normality test with ShapiroWilk test, we see that the data values for these variables are normally distributed.

For model-1, we find p-values as $p = 0.823$ for average total time, and $p = 0.757$ for average waiting time. For the resource utilization values, we have p-values as $p = 0.216$ for service technician and $p = 0.877$ for supply technician. Figure 6.22 shows the t-test results for model-1.

For model-2, we find p-values as $p = 0.581$ for average total time, and $p = 0.875$ for average waiting time. For the resource utilization values, we have p-values as $p = 0.938$ for acquirer, $p = 0.596$ for telecom supplier, and $p = 0.964$ for terminal supplier. Figure 6.23 shows the t-test results for model-2.

As a result, we strongly reject the null hypothesis due to high p-values and conclude that the results of the Arena model and the DEVSDSOL model are similar.

6.6.3. Model continuity in the example cases

The case examples showed that model continuity between the different models of the M&S lifecycle is obtained when the MDD4MS framework is applied successfully. When we look at the definition of model continuity in Section 1.4, we identify two main requirements for providing model continuity in a development process. These are transforming the initial and intermediate models, and preserving the modeling relation during the transformations. Hence, performing effective and successful model transformations in an MDD4MS process can ensure that model continuity is obtained.

We already presented the criteria for model transformations in Section 2.3.7. In the case example, the transformations satisfy the termination, uniqueness and readability requirements. Efficiency, maintainability, scalability and reusability requirements are partially supported since we made small scale examples and more experiments are needed for a better evaluation. Due to the fact that the model editors guarantee correct models, accuracy and consistency are implicitly guaranteed. Robustness is ensured by the model transformation language compiler. However, to consider a model transformation effective and successful, we pay attention to the correctness and completeness during the analysis. We focus on the following three aspects:

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
NumberOut_Repaired	Equal variances assumed	,357	,552	,706	58	,483	2,40000	3,39718	-4,40020	9,20020
	Equal variances not assumed			,706	57,961	,483	2,40000	3,39718	-4,40030	9,20030
NumberOut_NotRepaired	Equal variances assumed	,386	,537	-,724	58	,472	-2,46667	3,40674	-9,28600	4,35267
	Equal variances not assumed			-,724	57,987	,472	-2,46667	3,40674	-9,28604	4,35270
AvgTotalTime	Equal variances assumed	2,220	,142	,224	58	,823	,0005067	,0022578	-,0040128	,0050261
	Equal variances not assumed			,224	55,032	,823	,0005067	,0022578	-,0040180	,0050313
AvgWaitingTime	Equal variances assumed	2,159	,147	,311	58	,757	,0001833	,0005902	-,0009980	,0013646
	Equal variances not assumed			,311	52,113	,757	,0001833	,0005902	-,0010008	,0013675
ServiceTechnician_Utilization	Equal variances assumed	,000	1,000	-1,252	58	,216	-,0007833	,0006258	-,0020361	,0004694
	Equal variances not assumed			-1,252	57,570	,216	-,0007833	,0006258	-,0020363	,0004696
SupplyTechnician_Utilization	Equal variances assumed	2,435	,124	,155	58	,877	,0001000	,0006452	-,0011915	,0013915
	Equal variances not assumed			,155	55,675	,877	,0001000	,0006452	-,0011927	,0013927

Figure 6.22: T-test result for Arena and DEVSDSOL models for model-1.

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
AvgTotalTime	Equal variances assumed	,030	,862	-,556	48	,581	-,0060440	,0108651	-,0278897	,0158017
	Equal variances not assumed			-,556	47,739	,581	-,0060440	,0108651	-,0278928	,0158048
AvgWaitingTime	Equal variances assumed	,262	,611	-,159	48	,875	-,0016360	,0103067	-,0223590	,0190870
	Equal variances not assumed			-,159	46,648	,875	-,0016360	,0103067	-,0223746	,0191026
Acquirer_Utilization	Equal variances assumed	5,366	,025	,078	48	,938	,0000840	,0010772	-,0020818	,0022498
	Equal variances not assumed			,078	37,514	,938	,0000840	,0010772	-,0020976	,0022656
TelecomSupplier_Utilization	Equal variances assumed	1,565	,217	,534	48	,596	,0007120	,0013330	-,0019682	,0033922
	Equal variances not assumed			,534	46,162	,596	,0007120	,0013330	-,0019710	,0033950
TerminalSupplier_Utilization	Equal variances assumed	2,478	,122	-,045	48	,964	-,0000520	,0011453	-,0023548	,0022508
	Equal variances not assumed			-,045	43,452	,964	-,0000520	,0011453	-,0023611	,0022571

Figure 6.23: T-test result for Arena and DEVSDSOL models for model-2.

- Syntactic correctness of the target model,
- Completeness of the model transformation,
- Semantic correctness of the model transformation.

Syntactic correctness of the target models

If the target model conforms to the target modeling language, then it is syntactically correct. Unfortunately, the language and the compiler that we used during the transformation rule writing does not have support to ensure the target model correctness. So, we have spent extra effort to ensure the syntactical correctness manually. For example, the following three rules will generate three different ID numbers for a component, whereas only hexadecimal numbers are accepted in the target metamodel.

```
ID <- '123'
ID <- 'abc123'
ID <- 'zzz123'
```

So, the last model will not be correct although it is possible to write the rule with ATL. However, we don't see this as a problem in the future due to the fact that tools can be improved to provide support for rule writing. As a result, by verifying each rule, we guarantee that for a correct source model a syntactically correct target model will be generated. Auto generated model editors that we used in the case example ensures that only correct models are showed on the screen. Hence, in both examples, syntactically correct models are generated in every stage.

Completeness of the model transformations

In order to analyze completeness, we will measure source metamodel coverage and target metamodel coverage metrics. Source metamodel coverage calculates the quotient between the total number of distinct classes from the source metamodel that are covered in the model transformation, and the total number of classes from the source metamodel [Vig09]. Target metamodel coverage calculates the quotient between the total number of distinct classes from the target metamodel that are used in the model transformation, and the total number of classes from the target metamodel [Vig09]. Source metamodel coverage guarantees that the transformation is applicable to every model of the source language. Target metamodel coverage is important to generate precise target models. In the case example, all of the ATL transformations cover fully the source metamodel and the target metamodel. For the Java to code transformation, we only provide source metamodel coverage but not the target metamodel coverage. Because, we use all of the concepts from the DEVDSOL library but not the whole Java programming language. Hence, the transformations provide completeness in order to preserve and reuse the information in the source model.

During the transformations, we ensure the label similarity [MKY06] between the source and the target models. We assume that the additions and extensions to the existing information do not cause information loss. For example, if a task name is 'ABC' and it is transformed into a component named 'taskABC' in the target model then we assume that the information is preserved. Because it is always possible to obtain the original name [EEE+07]. Due to the fact that our transformation rules are complete we expect that structure and information preservation will be satisfied.

Semantic correctness of the model transformation

If a model m_1 transforms into m_2 then m_2 needs to preserve the semantics of m_1 to guarantee semantic correctness as well as model continuity. Only continuous transformations are deemed useful and meaningful [MV11]. This can in principle be checked by executing the semantic mapping and comparing the results. In computer science, there are various methods to check semantics preservation such as trace equivalence, bisimulation and behavioral equivalence.

In this research, we assume trace semantics for process models and DEVS models, and test trace equivalence between BPMN and DEVS models. In this case, the behavior of a process model or a DEVS model is a set of traces. A trace of a model m refers to one of its possible executions. A trace is an ordered list of labels representing the time-ordered events occurring in the execution of the model. Two models are trace equivalent if and only if they produce equivalent sets of traces.

To express the behavior of the sample models we will simplify the models for clarity. Figure 6.24 shows the simplified models for model-1. Although we use two Start elements in the original model we group them into one element in the simplified model. In the same way, we group the End elements into one End element as well. Serial tasks are also grouped into one element regardless their type.

Let $Tr(m)$ denotes the set of all traces of a model m, where an element of $Tr(m)$ is called a trace of m [NV09]. Two models m_1 and m_2 are said to be trace equivalent if and only if $Tr(m_1) \cong Tr(m_2)$. Based on the work of Burch et al. [BPSV03], we assume that if $Tr(m_1) \subseteq Tr(m_2)$ then $Tr(m_1) \cong Tr(m_2)$. During the model transformation from BPMN to DEVS, a mapping from each part of a BPMN model to one or more DEVS modeling elements is guaranteed. In this way, we ensure that $Tr(m_{bpmn}) \subset Tr(m_{devs})$. Based on our assumption, we conclude that $Tr(m_{bpmn}) \cong Tr(m_{devs})$. For example, in Figure 6.24, it is observable that $Tr(m_{bpmn}) \subset Tr(m_{devs})$.

The BPMN and DEVS models for the payment terminal application process can also be proven to be trace equivalent in the same way. Figure 6.25 shows the simplified models for model-2. In both examples, we preserve the structure and the existing information while moving from BPMN to DEVS.

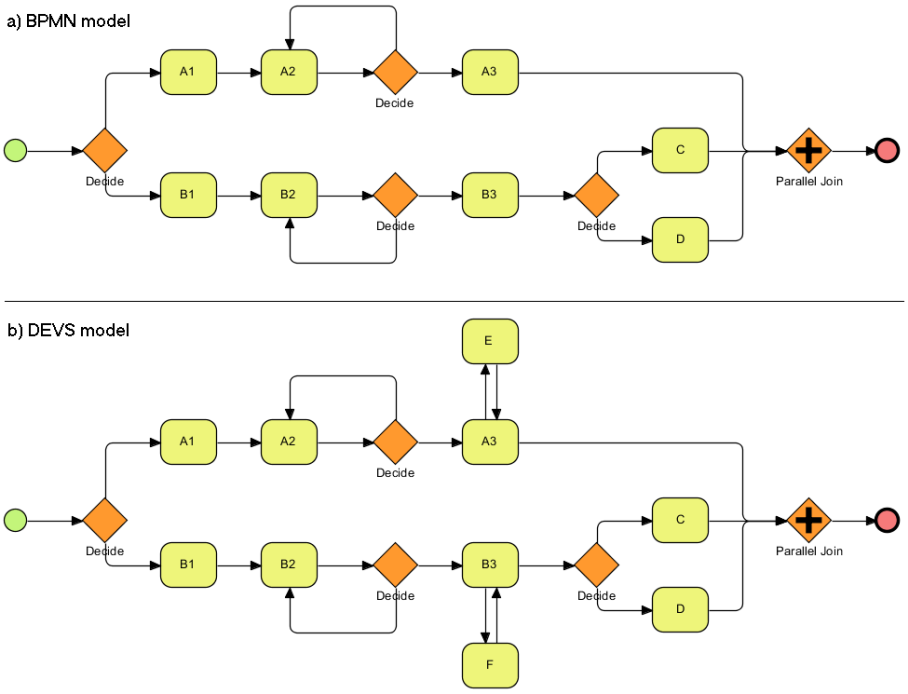


Figure 6.24: Simplified BPMN and DEVS models for the customer service process.

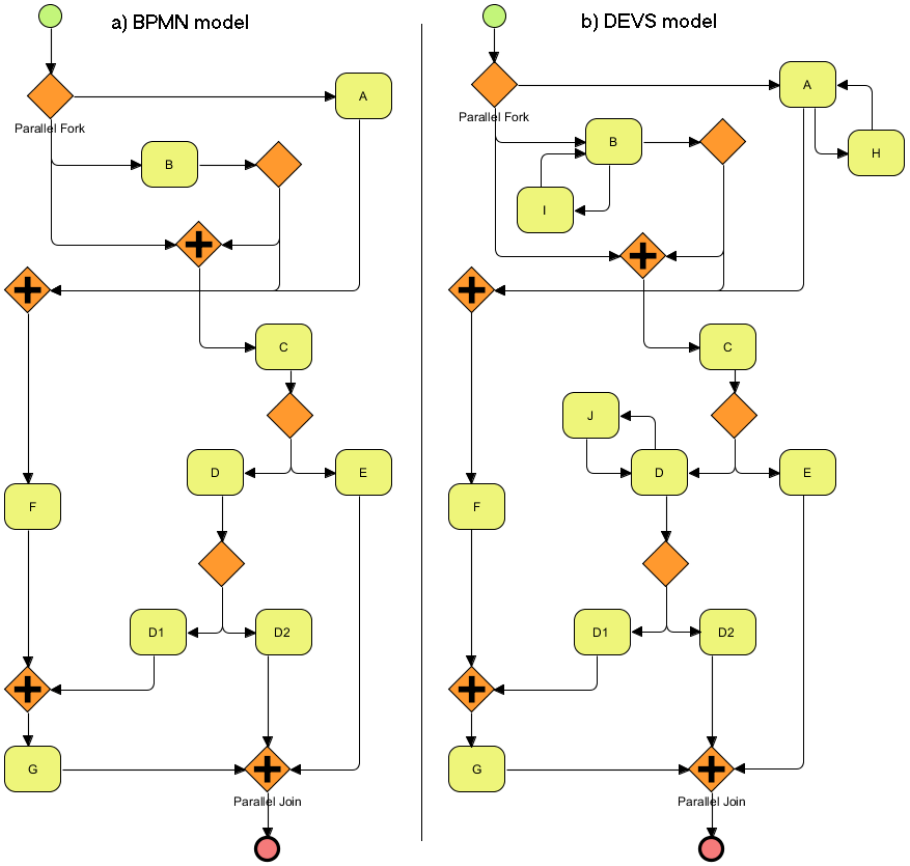


Figure 6.25: Simplified BPMN and DEVS models for the payment terminal application process.

For more complex business process models, a recent study of Kunze et al. [KWW11b, KWW11a] provides a metric based evaluation method for behavioral similarity. They use a 'behavioral profile' definition to express the behavior of a model. A behavioral profile captures behavioral characteristics of a model by three relations between pairs of activity nodes. These relations are based on the notion of weak order. Two activities of a process model are in weak order, if there exists a trace in which one activity occurs after the other. A pair $(x, y) \in A \times A$, where A is the finite non-empty set of activity nodes, is in one of the following relations: the strict order relation, the exclusiveness relation and the interleaving order relation.

The transformation of the DEVS components into JAVA classes are performed according to the DEVS operational semantics. By using the validated DEVSDSOL classes we ensure that the semantics of the DEVS model is preserved in the JAVA model. During the code generation a one to one mapping from each part of a Java model to Java source code is guaranteed. As a result, our transformations are effective and successful, and model continuity is obtained in both examples.

6.6.4. Satisfying the requirements for conceptual modeling

Lastly, we evaluate the case study according to the conceptual modeling requirements given in Section 2.1.1. The presented case study satisfies all of the requirements for conceptual modeling as:

- **R-CM.1.** The business process modeling domain ontology is described with the BPMN metamodel.
- **R-CM.2.** BPMN is used for conceptual modeling,
- **R-CM.3.** The auto generated model editor ensures that any conceptual model conforms to BPMN.
- **R-CM.4.** The system structure and abstract behavior are defined in the BPMN models,
- **R-CM.5.** The boundaries are defined in the BPMN models,
- **R-CM.6.** The BPMN models are communicative between the stakeholders,
- **R-CM.7.** The BPMN models are independent from the implementation details.

When we look at the requirements for conceptual modeling, we see that the first three requirements are related to the domain ontology and the conceptual modeling language rather than the conceptual model itself. Metamodeling provides a sound method for specifying modeling languages as well as ontologies. In case of using a domain specific language via a metamodel within the MDD4MS framework, the first three requirements are satisfied such that:

- **R-CM.1.** The problem/research domain ontology is described with the metamodel due to the fact that a valid metamodel shows an ontology.
- **R-CM.2.** The modeling language which the metamodel represents is used for conceptual modeling,
- **R-CM.3.** Auto generated model editor ensures that the conceptual model conforms to the modeling language.

During the case study, it is shown that these requirements are guaranteed by the BPMN metamodel. The major requirement for metamodeling is specifying all or the core parts of the language specification according to the modeling needs. We have chosen the core elements of BPMN and the metamodel is fully compatible with the BPMN specification version 2.0 [OMG11a]. Besides, the conceptual models, which are the instances of the BPMN metamodel, are transformed into the DEVS models and effectively used in the further steps. As a result, the MDD4MS framework supports and improves the conceptual modeling stage.

Chapter 7

Epilogue

This chapter presents the conclusions, the research findings and the future work. Before drawing the final conclusions, we present a summary of our research and how we satisfy the necessary requirements throughout this thesis in Table 7.1. There are different aspects of this research and the contributions are mainly in the field of simulation conceptual modeling, model driven development, component based simulation and business process modeling.

7.1. Conclusions

MDD approaches place models in the core of the entire system development process. They provide better and faster ways of developing systems through automated model transformations between models which are specified with well-defined modeling languages. Applying MDD in M&S provides new capabilities for efficient development of reliable, error-free and maintainable simulation models. MDD supports formal validation and verification techniques and provides early detection of the flaws. Availability of the existing tools and techniques for both metamodeling and model transformations is one of the practical advantages of MDD. Metamodeling provides a precise way for specifying the models and modeling languages. The most important feature of an MDD process is model continuity.

This research study proposes a comprehensive theoretical framework for model driven development of simulation models. The framework suggests three intermediate models on top of the rigid simulation model (i.e. the final executable simulation source code). Using intermediate models provides good understanding of the simulation model by different parties. We show that the proposed framework obtains model continuity via metamodel-based formal model-to-model transformations. The case study illustrates that the framework is applicable in the discrete event simulation domain.

MDD is different from the traditional development approaches and it requires a learning period and change of the programming habits. When the modeling languages are not available and the team members have little or no knowledge about MDD, it may take a large amount of time to develop metamodels. However, once

Table 7.1: Satisfying the requirements throughout the thesis.

Requirements	**How they are satisfied**
Requirements for the application of MDD (Section 2.3.8)	The MDD4MS framework applies the MDD concepts successfully and satisfies all of the requirements, namely abstraction, metamodeling, transformation, automation and generality (Section 3.6).
Requirements for simulation conceptual modeling (Section 2.1.1)	The presented case study satisfies all of the requirements for conceptual modeling. BPMN is used for conceptual modeling and the business process modeling domain ontology is described with the BPMN metamodel (Section 6.6.4).
Requirements for simulation model components (Section 5.2.2)	The DEVS components for BPMN modeling elements satisfy the modularity, interoperability, reusability, functionality, reachability and flexibility requirements. The upgradeability and replaceability requirements are not examined in detail due to time limitations of the research (Section C).
Requirements for providing model continuity (Definition 14 p.61)	The MDD4MS processes for the two example business process models satisfy the requirements for model continuity through automated formal model transformations (Section 6.6.3).
Requirements for a successful model transformation (Section 2.3.7)	The model transformations in the MDD4MS prototype ensure correctness, completeness and termination requirements (Section 2.3.7 and 6.6.3).

they are developed, further development time and costs could decrease significantly [TGS+05, BBG05, KJB+09, MCM13]. Working with metamodels is easier and beneficial as soon as it is understood well. Although traditional systems modeling and software engineering approaches can be chosen in small-scale and short-term projects, the model driven approach is more desirable for large-scale and critical simulation projects.

From the initial research questions and the final research evaluation, we conclude that this research addresses the identified issues in current M&S practice and theory. The applicability of the proposed framework is tested via prototyping. The proposed MDD4MS framework, performed case study and the example models show that the MDD approach can be successfully applied into simulation and the MDD4MS framework bridges the gap between simulation conceptual modeling and simulation model development stages in the M&S life cycle.

The outcomes of this research can be summarized as follows and the deliverables can be found in the MDD4MS project website [Çet13]):

- The MDD4MS framework which provides a set of methods and guidelines.
- A metamodel for BPMN
- A metamodel for hierarchical DEVS
- A simplified metamodel for JAVA
- A modeling editor for BPMN modeling
- A modeling editor for DEVS modeling
- A modeling editor for JAVA modeling
- An extensible conceptual modeling method for simulation (SimCoML)
- A metamodel for SimCoML
- A modeling editor for SimCoML
- A model transformation method from BPMN to DEVS
- A model transformation method from DEVS to JAVA model
- A model transformation method from JAVA model to JAVA code
- A DEVS component library for BPMN
- Sample case-1: Customer service process (bpmn, devs, java models and executable code)
- Sample case-2: Payment terminal application process (bpmn, devs, java models and executable code)
- Publications, tutorials, a user guide and a developers guide

7.2. Answers to the research questions

In this section, we revise the research questions given in Section 1.5. The first question is about supporting the conceptual modeling stage in M&S. To answer the first research question, two subquestions have been introduced. First, the requirements for an effective conceptual modeling stage in M&S are identified and then it is analyzed how a conceptual modeling language can help to meet these requirements. The first subquestion is answered in Section 2.1 during the background research by listing seven major requirements. The second subquestion is answered by applying a metamodeling approach at the simulation conceptual modeling stage. Section 2.3.2 presents some background information about metamodeling, and Section 3.2 and 3.3.2 explains how metamodeling can be used in an MDD process. Section 4.3 illustrates an extensible conceptual modeling metamodel for simulation to illustrate how a metamodel can be developed and used for a DSL. Besides, Chapter 6 presents a case study to show how a conceptual modeling language can guarantee three of the seven requirements for conceptual modeling.

The second main question is about providing model continuity throughout the M&S lifecycle. Again, to answer the second research question, two subquestions have been introduced. First, a method to utilize the simulation conceptual models in the further steps of the simulation study is proposed and then it is analyzed how formal model transformations can help to bridge the gap between the different models in the M&S lifecycle. The first subquestion is answered by defining and using metamodel based formal model transformations to transform simulation conceptual models into more detailed new models. The new models are expected to preserve the information in the conceptual model as well as include more detail. Section 2.3.5 presents some background information about model transformations, and Section 3.2 and 3.3.3 explains how formal model transformations can be used in an MDD process. Section 6.5.4 explains how simulation conceptual models can be transformed into other models in a practical case. The overall process by using metamodels and formal model transformations provides answer to the second subquestion. Chapter 3 presents the main conceptual framework, followed with two supporting approaches as using DSLs (in Chapter 4) and using simulation model components (in Chapter 5). Chapter 6 presents a practical example to show how model continuity is obtained throughout the M&S lifecycle.

As a result, we answer all of the research questions. The objective of this research is stated in Section 1.5 as follows: To design a framework for M&S that would provide a set of methods and guidelines for specifying (conceptual) models in a well-defined manner (to address issue 1), for performing formal model transformations on those models (to address issue 2), and for supporting model continuity throughout the M&S lifecycle (to address issue 3). Analyzing the research questions and the answers provided, we can state that we have accomplished this objective with this research. As a result, this research study ends up by proposing formal and practical solutions to the identified issues in Section 1.4. Our hypothesis that is formulated at the beginning of the research (as the use of the MDD methods, techniques

and tools can improve the conceptual modeling stage in simulation studies and provide model continuity between the different models in the M&S lifecycle) is tested throughout the research and it is supported by the results.

7.3. Research findings and reflections

It is important to follow an iterative process during the scientific research so that the relevance of the solution can be decided early in the research time line. In this research, initial ideas are presented at the simulation conferences and an early prototype with GME is developed at the beginning of the research [ÇVS10a]. By the help of this early prototype, the advantages of the MDD approach are observed at the first year of the PhD research. Reflecting on the results of the first prototype evaluation, the MDD concepts are applied into the M&S field to address the identified issues. After that, the conceptual framework is proposed in detail and formalized as well as a new prototype with Eclipse is developed.

Throughout this research, we have observed the following advantages of the MDD approach:

- Improved communication and information sharing via models at different abstraction levels (Section 6.5.8),
- Automation during the generation of model editors and source code (Section 6.5),
- Improved software modularity and consistency with the use of component based approach during code generation (Section 5.3),

On the other side, we have observed the following disadvantages and challenges:

- Model transformation languages and tools that we used are not practical and easy to use,
- Transformation rule writing needs experience and it is costly.

Potential users of the research results are all the actors in the simulation model development process, including conceptual modelers, simulation modelers and simulation analysts. This research provides new insights in modeling and simulation field and it aims at improving the conceptual modeling stage and increasing the reuse of simulation model components in M&S. The focus on the conceptual modeling stage and the application of MDD concepts into the simulation field to effectively use the conceptual models grant the originality of this PhD research, due to fact that this subject has not been adequately studied yet in the simulation field. The proposed generic MDD4MS framework can ensure model continuity by the use of the gained insights about conceptual modeling.

If we look at the definition of a conceptual model (see page 19), we identify two main properties as: a conceptual model is an abstract representation and it is not executable. However, the overall objective of this research is to increase the level of abstraction via model transformations and to obtain an executable model. Hence, at each transformation, we add new data or information to the model so that we have new models at a different abstraction level. As well as, we identify an important requirement for conceptual models as (see page 22): a conceptual model must be independent from the implementation details. This means that a conceptual model is both implementation independent and platform independent. However, it is not easy to say that it is paradigm independent as well with the current available conceptual modeling languages.

The MDD4MS framework provides practical and formal guidance for moving from a conceptual model to an executable simulation model. It improves conceptual modeling stage and provides model continuity during simulation model development. A possible reflection on systems engineering can be to utilize MDD4MS for model driven systems engineering. There can be two ways to adopt MDD for systems development. One of them is applying the concepts to whole systems engineering lifecycle not necessarily M&S is a part of the project. This way can be categorized under the MDE literature. The second way is using M&S during the design stage and implementing the results in the further stages. This is generally referred as M&S based design or M&S based systems engineering. In both cases, the MDD4MS framework can be adapted to the systems development process.

This research has been a good practice for component based simulation as well. Once the components were developed, validated and made available for reuse, they have improved the model transformations and fully executable models are generated. Component based approach also helped us to obtain consistent models and accurate results in a cost effective manner.

The transformation of the BPMN elements into DEVS components has provided an effective way to easily model and simulate business processes. Because, modelers not only need graphical presentations and animations during the simulation but also require accurate and correct simulation results. Hence, formalizing the steps and transforming the BPMN elements into mathematically sound DEVS components provides a formal approach for business process simulation.

As a final remark, the metamodels, the visual editors and the code generator provide a higher layer tool architecture on top of the DSOL simulation suite and enhance DSOL with a graphical user interface. Besides, the DEVS metamodel is used as the PISM metamodel in the MDD4MS framework. Thus, the generated DEVS models are free from the implementation details and they can be transformed into different platform specific models for various DEVS simulation platforms. So, the new approach will help the modelers to construct their simulation models faster, better and more reliable.

7.4. Further research

As shown in the case example, MDD4MS framework is applicable in the DEVS-based discrete event simulation domain. MDD4MS presents a generic framework and it is also applicable in different domains. Due to the fact that there are related studies on applying MDD and/or component based approach in simulation field (see Section 2.4.2), we believe that adapting them to the MDD4MS framework can formalize the existing applications. Both to evaluate the MDD4MS framework and to validate the MDD4MS prototype we had small scale cases and experiments. We need more examples on the larger scale and the future work will include using the MDD4MS framework in a large scale real life M&S study.

Agent based simulation, HLA-based distributed simulation, PetriNets-based simulation, SysML-based simulation, etc. can be future implementation areas for the MDD4MS framework. For example, Ogston and Brazier [OB11] propose to use generic interfaces for platform-independent experimental data during multi agent systems development and advocates a comprehensive development cycle. As a future work, adaption of the MDD4MS framework to multi agent systems development can help to formalize the steps and better categorize the M&S and systems engineering processes.

As well as, an important future research topic is about component composability and interoperability. As stated in Chapter 5, the simulation model components are usually platform dependent and not compatible with other components developed in different environments. MDD has been shown to be very effective and useful to perform successful compositions. However, more research needs to be done to fully elaborate the mathematical foundations of M&S Science and to show the advantages of higher level conceptual and platform-independent models [TDPHZ13].

During the analysis stage, we have identified that the M&S literature is lacking the methods and metrics to measure model continuity. We believe that the theoretical computer science methods can be utilized for further analysis of conceptual models, simulation models and semantics preservation during the transformations. A possible future work will define formal analysis methods and metrics to evaluate the MDD4MS processes for model continuity as well as other properties. Transformation validation and verification methods in the MDD literature can help to implement automated solutions for more detailed analysis of transformations.

Appendix A

Basic Definitions for the Frequently Used Terms

Due to inconsistent terminology in the literature, we give some basic definitions of the frequently used terms to provide a common understanding. These definitions are highly influenced by the software engineering and the systems engineering body of knowledge [ISO10, PMI08].

Abstraction level: level of detail from different perspectives or aspects.

Discipline: a branch of scientific knowledge, field of study.

Framework: a reusable method or a set of methods that can be refined (specialized) and extended to support a methodology.

Guideline: a statement that provides information about how to apply a method or a technique.

Language: a means of expressing or communicating in a structured way by using gestures, signs, symbols, letters, numerals, sounds, etc.

Lifecycle: evolution of a system, product, service, project or other human-made entity from conception through retirement.

Method: a systematic procedure that may be used to perform a process or a task and that may employ one or more techniques (e.g: conceptual modeling method, simulation model specification method, analysis method, etc.).

Methodology: a body of knowledge comprising the methods, techniques, procedures, approaches, guidelines, principles, patterns and/or tools that may be used to perform a process or a set of processes in a discipline or a particular domain (e.g: modeling and simulation methodology).

Phase: a collection of logically related activities in a lifecycle, usually culminating in the completion of a major deliverable.

Procedure: ordered series of steps that specify how to perform a task.

Process: a set of interrelated or interacting activities for a purpose.

Project: an endeavor with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements.

Project lifecycle: a collection of generally sequential and sometimes overlapping project phases whose name and number are determined by the control needs of the organization or organizations involved in the project. A life cycle can be documented with a methodology.

Stage: a collection of logically related sub-activities in a phase.

Step: a defined task in a stage that tells a user to perform an action (or actions).

Study:(or research study) a work that results from studious endeavor.

Technique: a systematic procedure that may be employed to perform a task and that may utilize one or more tool.

Tool: an instrument to perform some task

Appendix B

Introduction to Formal Language Theory

In this appendix, we provide a brief introduction to formal language theory. In formal language theory, a language is a set of expressions (or sentences) each finite in length and constructed from a finite set of symbols [Cho02]. A language consists of a syntax and semantics that each can be defined formally or informally. Formal language theory focuses on defining a formal syntax, which is often described by means of a grammar [Lin12]. The finite set of symbols used to construct the expressions is called the alphabet Σ. From the individual symbols, expressions are constructed by composition. The infinite set of all expressions, which can be obtained by composing zero or more symbols from Σ, is denoted as Σ^*. A language is defined as a subset of Σ^*. An expression e in a language l is denoted as $e \epsilon l$.

A grammar is a generative mechanism which can generate expressions by using a set of production rules. A grammar includes terminal and non-terminal symbols. Non-terminal symbols are called variables. A subset of the variables is called the start variables. A production rule consists of a left hand side (lhs) and a right hand side (rhs). It is denoted as $lhs \Rightarrow rhs$, where each side consists of a sequence of the terminal and non-terminal symbols. Starting from a start variable, expressions are obtained by applying a number of production rules consecutively until the expression consists only of terminal symbols. This is called the derivation of an expression from a start variable. The intermediary expressions are called productions, while the final expression is called a well-formed expression.

Definition 19. *A grammar g is defined as a quadruple $g = \{T, N, I, P\}$ where*

T is a finite alphabet (terminal symbols),

N is a finite set of variables (non-terminal symbols), where T and N are non-empty and disjoint sets,

I is a finite set of start variables, where $I \subseteq N$,

P is a finite set of production rules, where a production rule is an ordered pair

from $(T \cup N)^* \times (T \cup N)^*$.

A grammar produces syntactically correct expressions, regardless whether the expression is meaningful or not. The set of all expressions generated by g is called the formal language generated by g and is denoted by $l(g)$ [Lin12].

Definition 20. *For a given grammar* $g = \{T, N, I, P\}$,

$l(g) = \{e \in T^* | I \Rightarrow^* e\}$, *where* $\Rightarrow^*$ *is a derivation of expression* e.

If a language l has a formal syntax, an expression in l can be parsed according to the grammar of l by a parser and the expression can be *verified* to be grammatically correct. Although the formal language theory focuses on syntax, it is closely related to the formal semantics. If the semantics of a language is defined as a formal system, then it is called a formal semantics. Otherwise, it is called an informal semantics and a language with a formal syntax and an informal semantics is called a semi-formal language. In mathematical logic, model theory provides a way for defining structures that give meaning to the expressions of formal languages [CK12]. If a language l has a formal semantics, an expression in l can be interpreted according to the semantics of the language by an interpreter and the expression can be *validated* to be semantically correct.

Appendix C

The MDD4MS Prototype Implementation Details

C.1. DEVS components

This appendix provides information about the DEVS component library for BPMN. Pools, swimlanes and subprocesses are modeled as coupled models and automatically generated. So, they are not part of the library. BPMN library includes the elements shown in Figure C.1. We provide a separate queuing library as well, which can be used for basic queuing system modeling.

C.1.1. Start event

The Start event is an event and represents the arrival of entities. To mimic this behavior in a simulation model, this event is translated into an atomic DEVS model. This component generates entities based on a certain specification. It is similar to the Create module in the Arena simulation software.

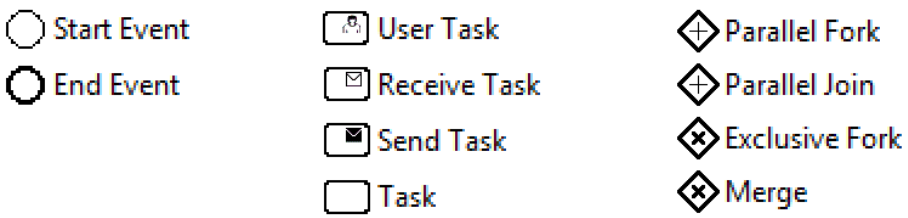


Figure C.1: Graphical representation of the BPMN elements used in the prototype.

The parameters describing start event specification are:

- *Entity type:* type of the entity that will be created (e.g., email, letter, order, customer, etc.)
- *Distribution mode:* the mode for the interarrival time distribution. Mode can be (1) constant, (2) exponential, (3) uniform, (4) triangular, or (5) normal distribution
- *Mean:* mean value of the distribution
- *Min.:* Minimum value for uniform or triangular distribution
- *Max.:* Maximum value for uniform or triangular distribution
- *stdDev:* Standard deviation for normal distribution

The formal description of the Start Event component is given by the state diagram as shown in Figure C.2. This component has one output port through which a newly created entity leaves, and one state, namely the 'Passive' state. The time duration after which an output function takes place is equal to the interarrival time as specified by the modeler. This interarrival time may be constant (e.g., every 10 minutes an entity is created), or following a statistical distribution (e.g., an entity is created on average after 5 to 10 minutes, following a uniform distribution). Based on the mode that is specified in the constructor, a different distribution is used for the random number generation of interarrival times. Each time an entity is generated, the sigma value (time remaining before the output function is called) is reset to the next inter-arrival time.

C.1.2. End event

The End event is an event and represents the end of a business process, namely when an entity leaves the system. To mimic the behavior of an entity leaving the system, the End event is formalized as an atomic DEVS model with two main states, namely 'Passive' and 'Active'. The component will remain in 'Passive' state, until an entity arrives at its input port, triggering an external transition. Then, the state changes to 'Active' and the entity is disposed. After that, the state changes back from 'Active' to 'Passive'.

An End event can be instantiated through its constructor. The constructor allows to specify the parent model, a name, a description and a unique identifier. An End event records several statistics about the entities and about the system. The state diagram of the End component is shown in Figure C.3.

C.1.3. User task

A User Task is an activity which represents work that is performed by a resource and spends a certain amount of time. To mimic the behavior of a resource performing

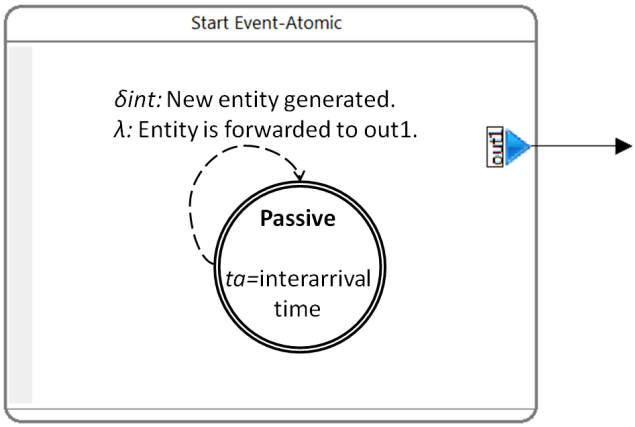


Figure C.2: State diagram for Start event.

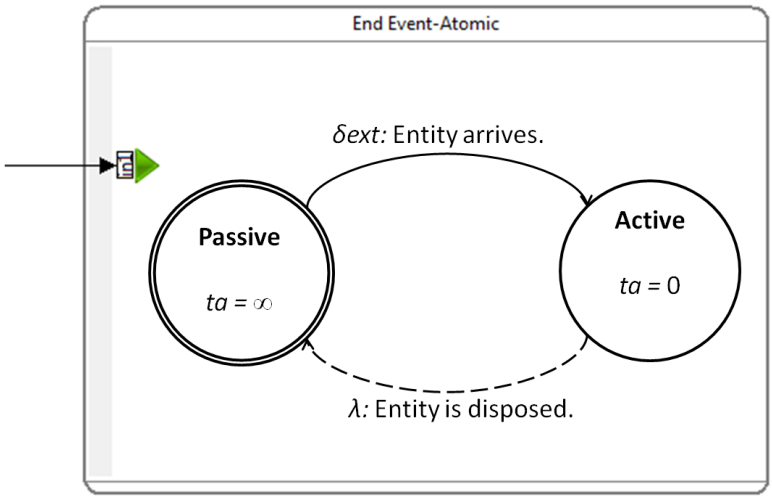


Figure C.3: State diagram for End event.

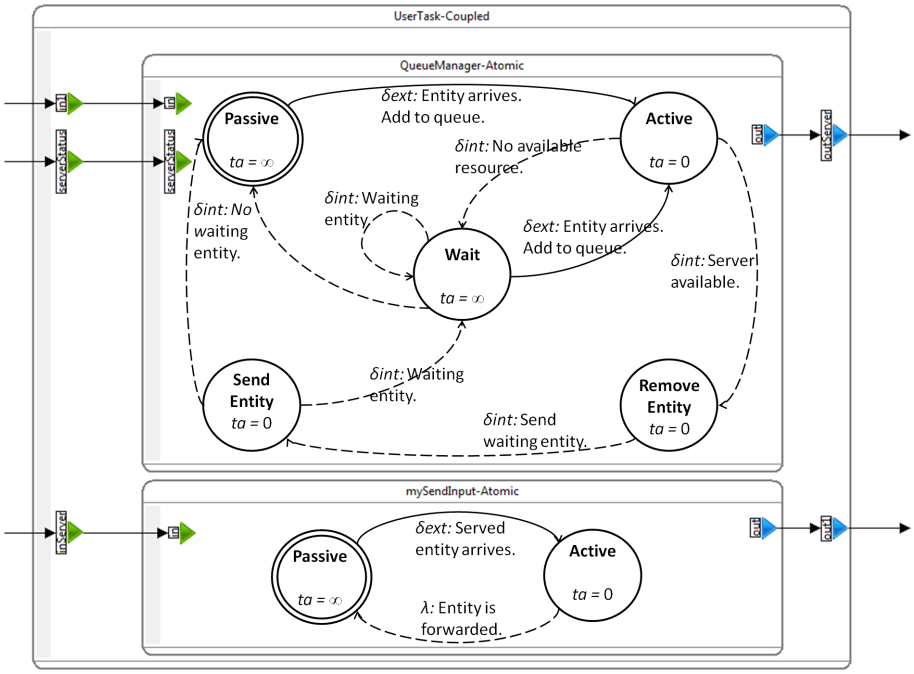


Figure C.4: Inner details of the User Task coupled component.

an activity for some time, the Task component delays an entity arriving at the input port for a specified duration before sending it to the output port. A task can be performed by multiple resources at the same time for different entities.

When a Task is in 'Passive' state, it means that no resource is currently seized. When an entity arrives, the state changes to 'Active' and a resource is allocated. After the certain amount of service time, which is defined by a distribution, the state changes to 'Passive' if no more resources are currently performing that task, or remains in 'Active' state if one or more resources are performing the same task. The user task utilizes a queue manager from a DEVS based queuing library. Queue capacity can be specified, but chosen 1000 as default. The user task component is implemented as a coupled model. The inner details of the user task component are shown in Figure C.4.

A User task is always linked to a server model (from the DEVS based queuing library) in a coupled model, where a server model represents a set of resources. The waiting time is calculated according to the resource availability and service rates. The service rate for a resource can be configured with mode, mean, min, max, and stdDev parameters as defined for Start event. Figure C.5 shows how

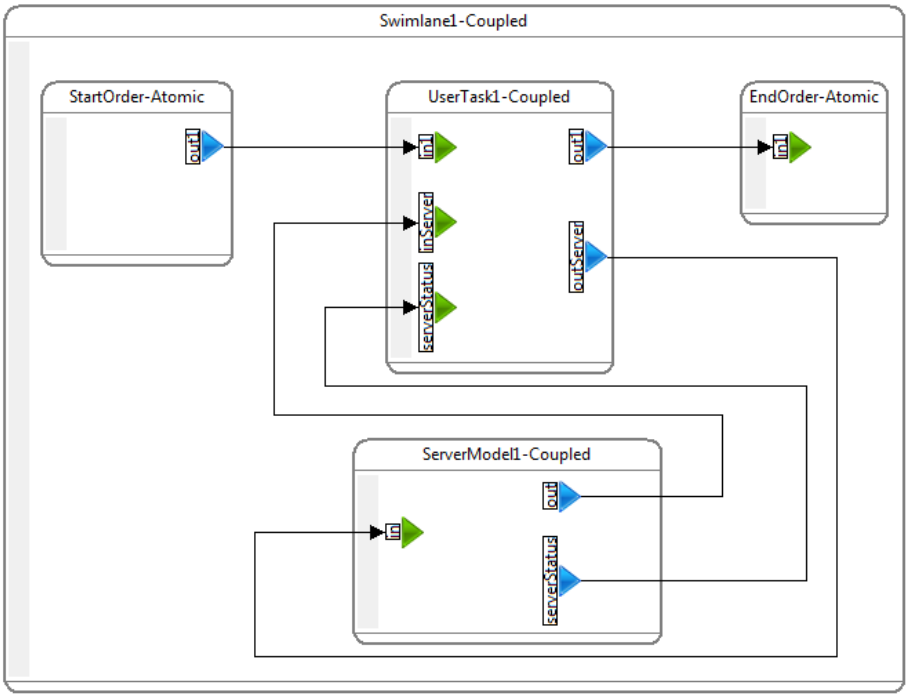


Figure C.5: Coupling User Task with a server model.

a user task component is coupled with a server model. The inner details of the server model component for three resources are shown in Figure C.6. Service rate and number of resources can be specified with parameters.

C.1.4. Simple task, Send task, Receive task

A Simple Task is an activity which represents some work that is performed at a certain amount of time. It is a simplified task component without a resource allocation mechanism. It can be compared to the Delay module in Arena. Send and receive tasks are special types of a simple task. They all have a default constant service time which can be changed. They are implemented as atomic models. The state diagram of the Simple Task component is shown in Figure C.7.

C.1.5. Exclusive fork and Exclusive join

An Exclusive Fork is a gateway which is used to represent decisions made in a business process and to direct the flow of an entity based on the evaluation of a condition. This condition can be either the evaluation of an entity specific attribute or probabilistic. Currently, an attribute based exclusive fork component can only

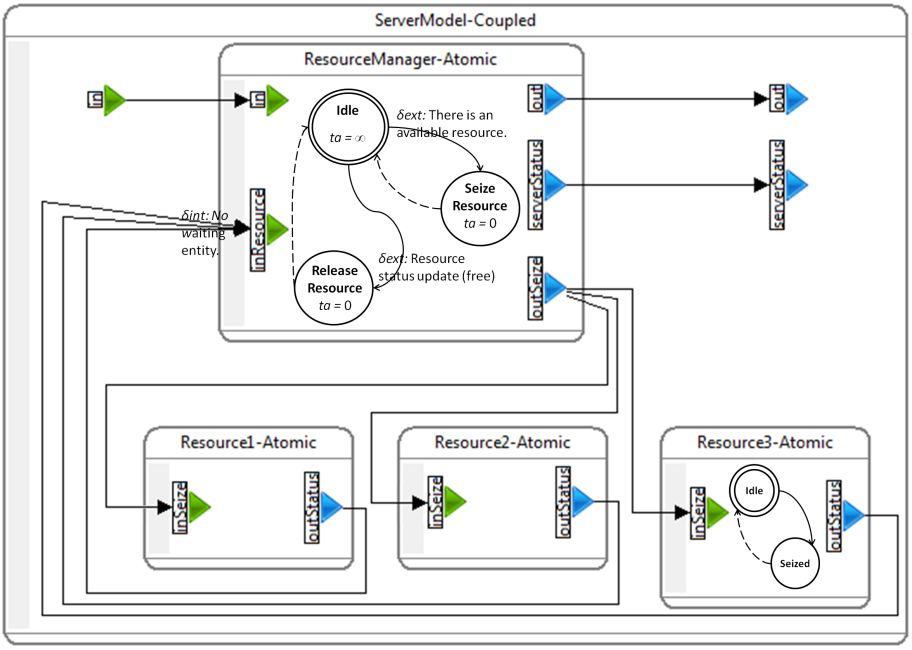


Figure C.6: Inner details of the Server Model coupled component.

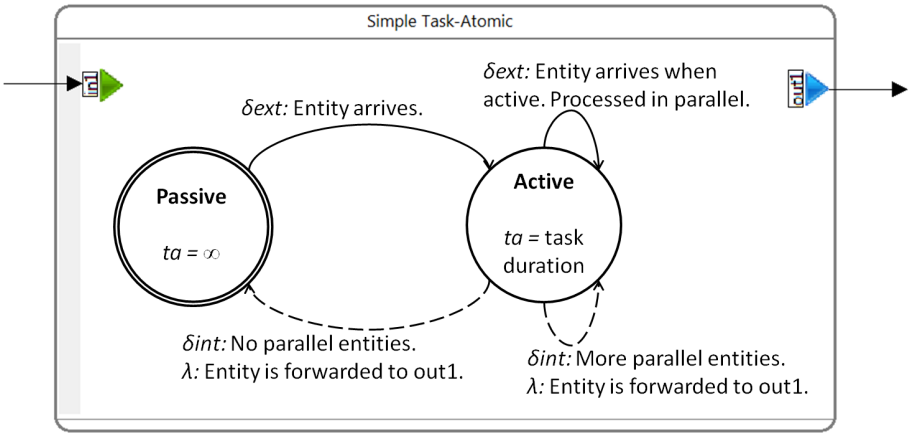


Figure C.7: State diagram for Simple Task.

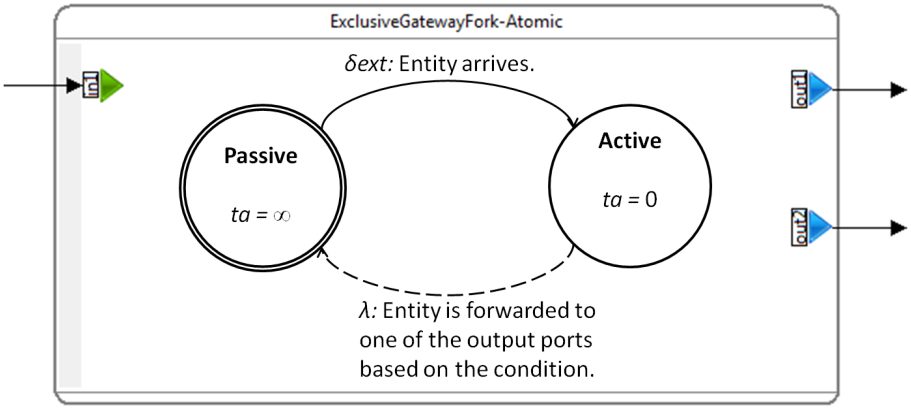


Figure C.8: State diagram for Exclusive Fork.

handle entity type attribute. The condition parameters should be defined by the modeler, otherwise a default 0.50 probability is used. In Figure C.8 the formalized DEVS model of an Exclusive Fork component is given. This component has one input port through which entities arrive and two output ports (out1 and out2) through which entities leave. It should be noted that an arriving entity can leave through only one output port not both output ports at the same time.

An Exclusive Join is a gateway which is used to merge coming flows and entities. It is implemented as an atomic model with two input ports and one output port. It stores the entities that comes to the input ports and merges them when both ports have entities. The state diagram of the Exclusive Join component is shown in Figure C.9.

C.1.6. Parallel fork and Parallel join

Parallel gateways are used to support modeling and simulation of parallel activities in a business process. A Parallel Fork duplicates an entity and sends the original to the out1 output port while it sends the duplicate to the out2 output port. The Parallel Fork component can be configured to have 3 or 4 output ports. In this case, the duplicates are sent to out3 and out4 ports. The state diagram of the Parallel Fork component is shown in Figure C.10.

A Parallel Join combines the original entity with its duplicates. Synchronization of parallel activities through a Parallel Join is based on the concept that an entity will wait in this gateway for an unspecified amount of time until its original (or another duplicate) entity arrives. The state diagram of the Parallel Join component is shown in Figure C.11.

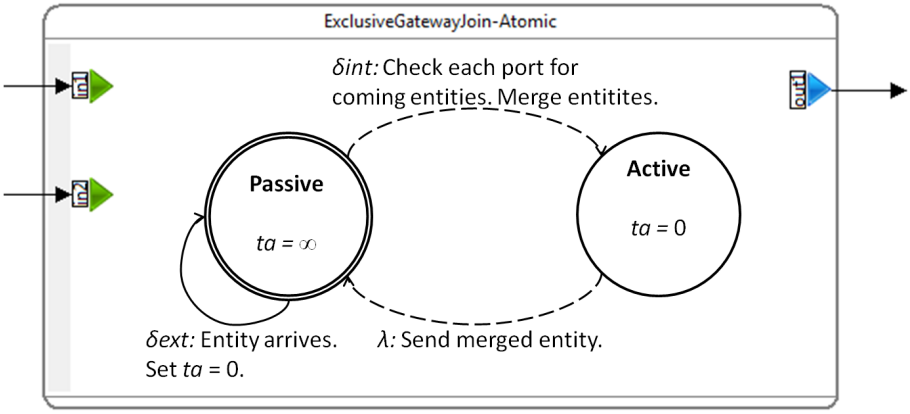


Figure C.9: State diagram for Exclusive Join.

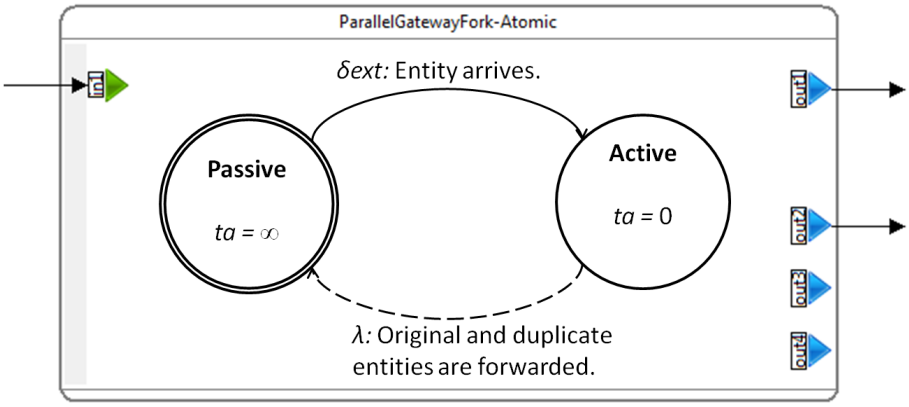


Figure C.10: State diagram for Parallel Fork.

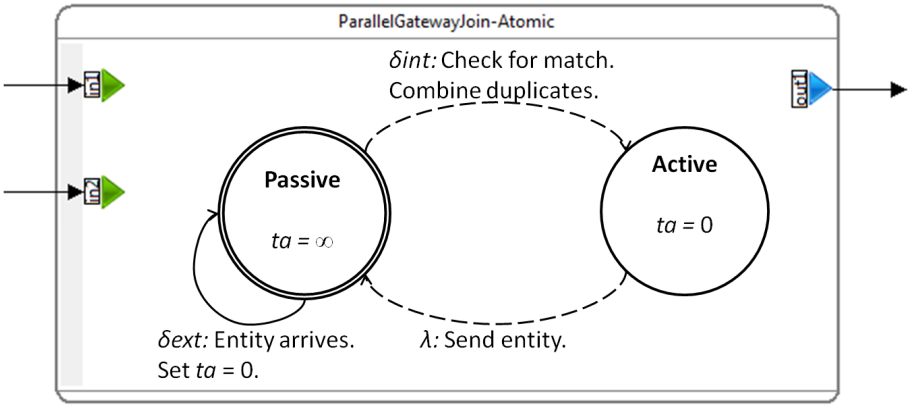


Figure C.11: State diagram for Parallel Join.

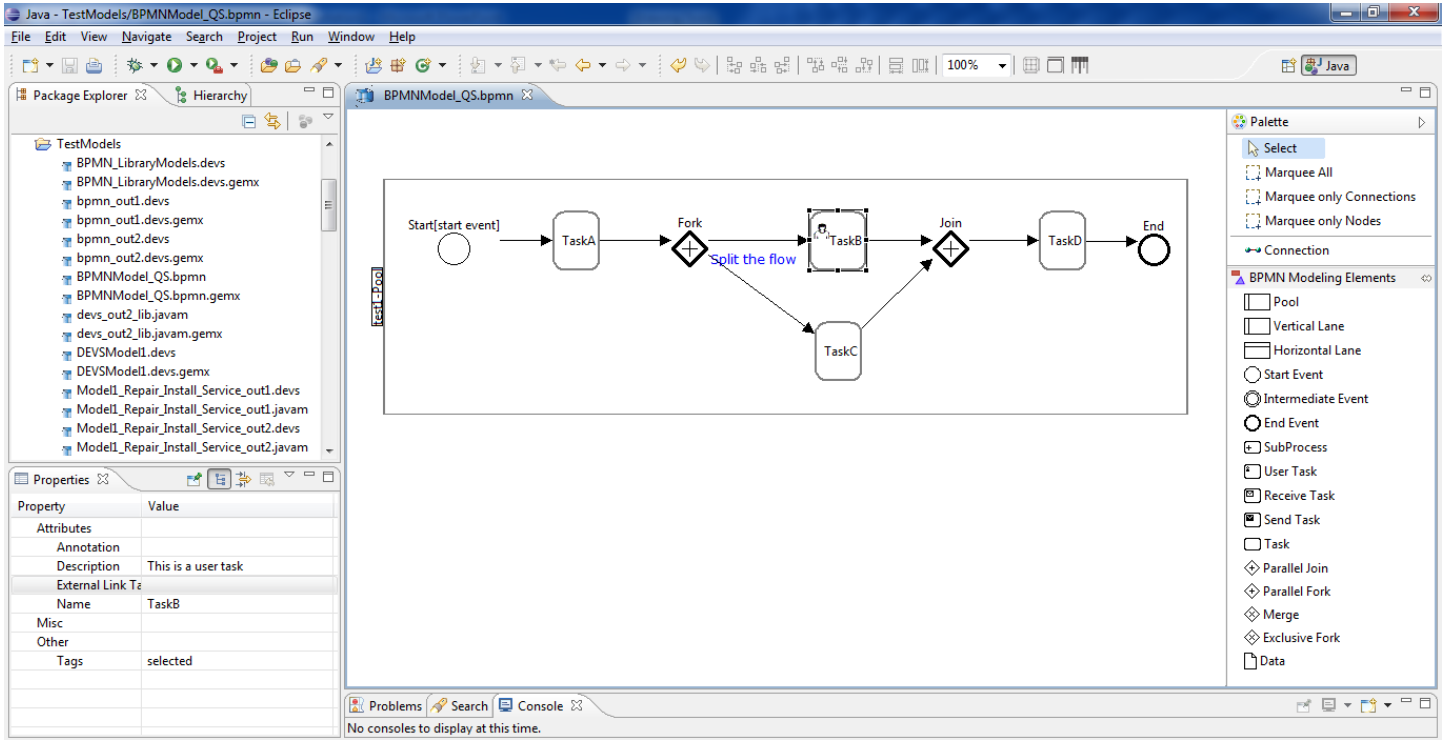


Figure C.12: BPMN model editor in the MDD4MS prototype.

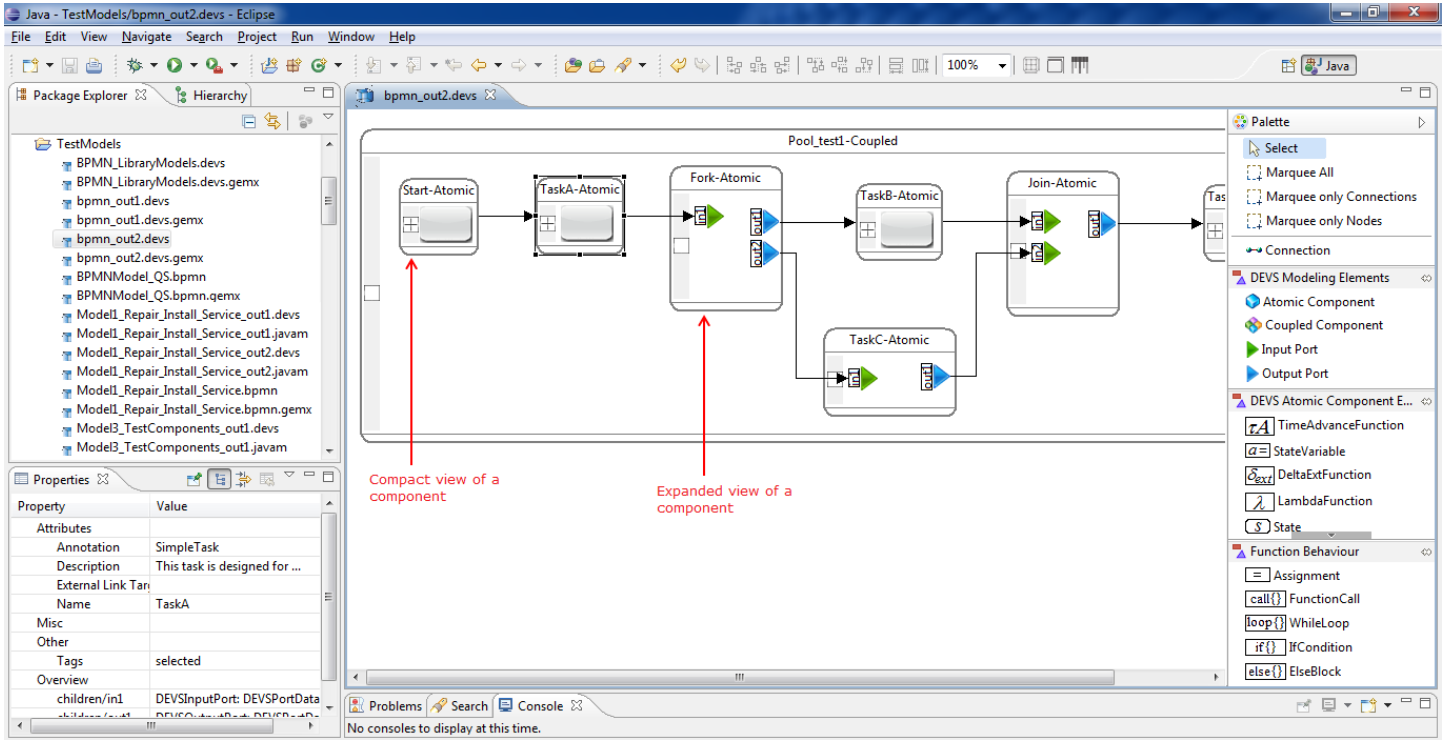


Figure C.13: DEVS model editor in the MDD4MS prototype.

Appendix D

Simulation Results for the Case Study

This appendix presents the experimental parameters and the results for the example models in Chapter 6. Model-1, i.e. the customer service process model, was executed for 30 replications (for both DEVS and Arena models). Each replication runs for 5000 hours with a 100 hours warm-up period. Model-2, i.e. the payment terminal application process model, was executed for 25 replications (for both DEVS and Arena models). Each replication runs for 2000 hours with a 50 hours warm-up period. The data for the average total time, average waiting time, and resource utilization has been collected. The similarity of the sample data sets are tested with t-test.

Table D.1: Experimental model and setup parameters for model-1.

$Parameter$	$Value$
$Setup$	
Run length	$5000\ hours$
Warm-up time	$100\ hours$
Number of replications	30
$Model$	
AtHome probability	0.95
NotDone probability	0.10
Arrival rate	$constant(2)$
Simple task duration	$constant(0.1)$
Service rate	$normalDist(1, 0.25)$
Appointment time	$normalDist(1.5, 0.2)$

Table D.2: Experimental model and setup parameters for model-2.

$Parameter$	$Value$
$Setup$	
Run length	$2000\ hours$
Warm-up time	$50\ hours$
Number of replications	25
$Model$	
Arrival rate	$constant(2.15)$
Simple task duration	$constant(0.1)$
Service rate	$normalDist(2, 0.25)$

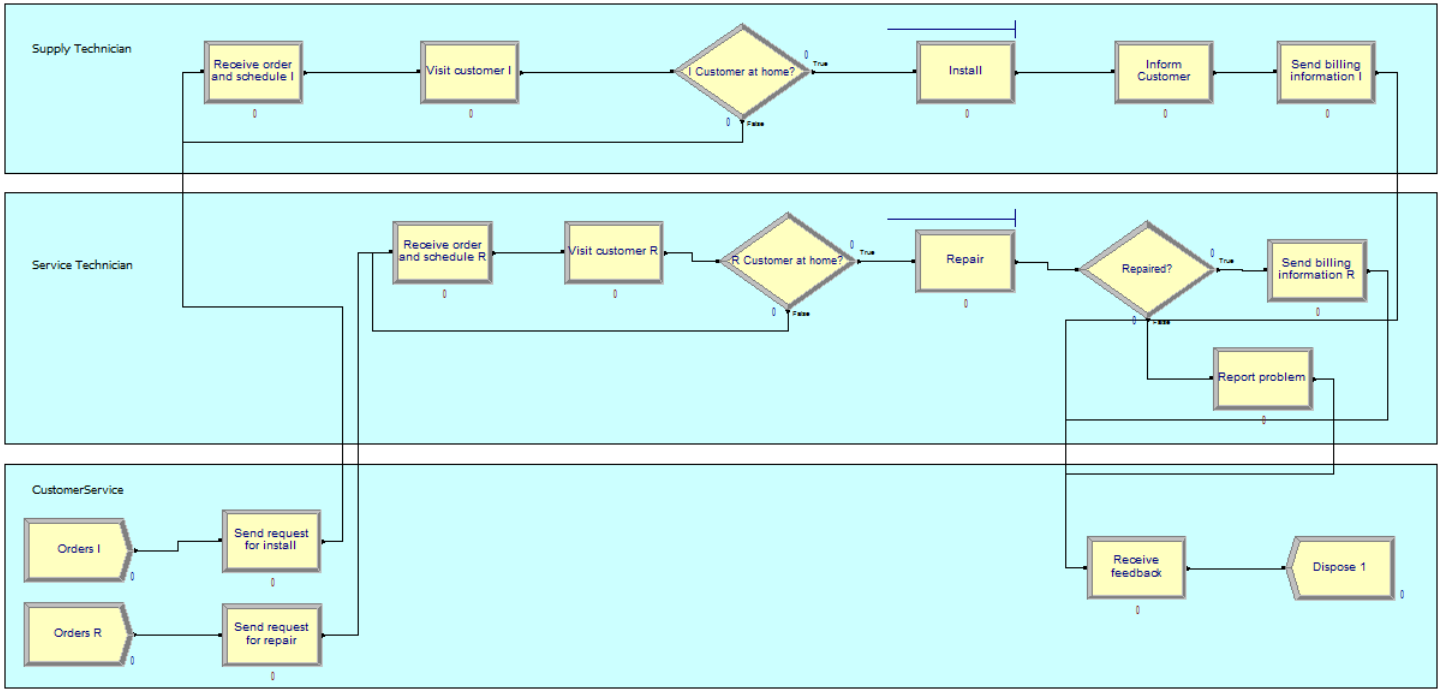


Figure D.1: Arena simulation model for the customer service process model (model-1).

	Group	NumberOut_Installed	NumberOut_Repaired	NumberOut_NotRepaired	AvgTotalTime	AvgWaitingTime	ServiceTechnician_Utilization	SupplyTechnician_Utilization
1	arena	2450,00	2201,00	249,00	3,1573	,0308	,5023	,5018
2	arena	2451,00	2240,00	210,00	3,1559	,0308	,5020	,5066
3	arena	2450,00	2213,00	237,00	3,1602	,0349	,5017	,5014
4	arena	2450,00	2217,00	233,00	3,1535	,0299	,5062	,5040
5	arena	2450,00	2202,00	248,00	3,1466	,0297	,4980	,4968
6	arena	2450,00	2212,00	238,00	3,1482	,0274	,5022	,5010
7	arena	2450,00	2213,00	237,00	3,1615	,0322	,5031	,4990
8	arena	2450,00	2224,00	226,00	3,1437	,0290	,5012	,5002
9	arena	2450,00	2211,00	239,00	3,1412	,0279	,4997	,4972
10	arena	2451,00	2198,00	252,00	3,1450	,0297	,4997	,4996
11	arena	2450,00	2179,00	271,00	3,1533	,0278	,4988	,5031
12	arena	2450,00	2182,00	268,00	3,1552	,0317	,5032	,4988
13	arena	2450,00	2202,00	248,00	3,1375	,0243	,4992	,5003
14	arena	2450,00	2208,00	242,00	3,1500	,0298	,4997	,4996
15	arena	2450,00	2207,00	243,00	3,1414	,0301	,5005	,5014

Figure D.2: Arena simulation results for model-1: Replications 1 to 15.

	Group	NumberOut_Installed	NumberOut_Repaired	NumberOut_NotRepaired	AvgTotalTime	AvgWaitingTime	ServiceTechnician_Utilization	SupplyTechnician_Utilization
16	arena	2450,00	2196,00	254,00	3,1518	,0301	,5028	,4994
17	arena	2450,00	2187,00	263,00	3,1557	,0311	,5013	,4993
18	arena	2450,00	2180,00	270,00	3,1356	,0237	,4979	,4990
19	arena	2450,00	2190,00	259,00	3,1450	,0326	,5019	,4948
20	arena	2450,00	2206,00	244,00	3,1454	,0280	,5013	,5023
21	arena	2450,00	2208,00	242,00	3,1377	,0252	,5030	,5004
22	arena	2450,00	2213,00	237,00	3,1478	,0296	,4978	,5019
23	arena	2450,00	2206,00	244,00	3,1616	,0334	,5042	,4997
24	arena	2450,00	2197,00	253,00	3,1509	,0280	,5035	,5019
25	arena	2450,00	2210,00	240,00	3,1391	,0254	,5015	,4997
26	arena	2450,00	2206,00	244,00	3,1596	,0312	,4974	,5025
27	arena	2450,00	2185,00	265,00	3,1466	,0310	,4963	,5014
28	arena	2450,00	2204,00	246,00	3,1601	,0309	,5035	,5013
29	arena	2450,00	2210,00	240,00	3,1460	,0289	,5025	,5017
30	arena	2450,00	2193,00	257,00	3,1549	,0329	,4990	,5010

Figure D.3: Arena simulation results for model-1: Replications 16 to 30.

	Group	NumberOut_Installed	NumberOut_Repaired	NumberOut_NotRepaired	AvgTotalTime	AvgWaitingTime	ServiceTechnician_Utilization	SupplyTechnician_Utilization
1	devsdsol	2450,00	2221,00	228,00	3,1429	,0302	,4964	,4983
2	devsdsol	2450,00	2209,00	241,00	3,1618	,0330	,5005	,5041
3	devsdsol	2449,00	2191,00	258,00	3,1517	,0308	,4977	,4977
4	devsdsol	2450,00	2211,00	239,00	3,1507	,0291	,4968	,5026
5	devsdsol	2450,00	2221,00	228,00	3,1479	,0303	,5010	,4998
6	devsdsol	2450,00	2191,00	259,00	3,1659	,0323	,4994	,5059
7	devsdsol	2450,00	2220,00	230,00	3,1575	,0306	,5007	,5010
8	devsdsol	2449,00	2211,00	239,00	3,1555	,0286	,5014	,4988
9	devsdsol	2451,00	2217,00	234,00	3,1401	,0266	,5058	,5031
10	devsdsol	2449,00	2195,00	254,00	3,1656	,0330	,5020	,4971
11	devsdsol	2450,00	2217,00	233,00	3,1495	,0307	,5015	,4994
12	devsdsol	2450,00	2203,00	248,00	3,1490	,0340	,4979	,5001
13	devsdsol	2449,00	2199,00	251,00	3,1451	,0290	,4975	,5060
14	devsdsol	2449,00	2180,00	271,00	3,1395	,0270	,5019	,4967
15	devsdsol	2449,00	2191,00	258,00	3,1588	,0299	,5003	,5018

Figure D.4: DEVSDSOL simulation results for model-1: Replications 1 to 15.

	Group	NumberOut_Installed	NumberOut_Repaired	NumberOut_NotRepaired	AvgTotalTime	AvgWaitingTime	ServiceTechnician_Utilization	SupplyTechnician_Utilization
16	devsdsol	2450,00	2216,00	235,00	3,1641	,0287	,5005	,5046
17	devsdsol	2450,00	2211,00	239,00	3,1368	,0288	,5008	,4971
18	devsdsol	2450,00	2188,00	263,00	3,1526	,0272	,5016	,5007
19	devsdsol	2451,00	2189,00	262,00	3,1335	,0276	,5000	,4982
20	devsdsol	2449,00	2194,00	256,00	3,1564	,0296	,5058	,4978
21	devsdsol	2450,00	2209,00	241,00	3,1381	,0301	,5010	,4972
22	devsdsol	2450,00	2204,00	245,00	3,1443	,0284	,5003	,5017
23	devsdsol	2450,00	2230,00	220,00	3,1683	,0291	,5004	,5001
24	devsdsol	2451,00	2213,00	236,00	3,1590	,0328	,4982	,5015
25	devsdsol	2449,00	2186,00	264,00	3,1453	,0297	,4969	,5018
26	devsdsol	2451,00	2221,00	229,00	3,1406	,0279	,5038	,5026
27	devsdsol	2450,00	2213,00	236,00	3,1434	,0303	,4967	,5004
28	devsdsol	2450,00	2214,00	236,00	3,1391	,0308	,4972	,5014
29	devsdsol	2450,00	2195,00	254,00	3,1437	,0286	,4999	,4976
30	devsdsol	2450,00	2212,00	238,00	3,1568	,0288	,5040	,5050

Figure D.5: DEVSDSOL simulation results for model-1: Replications 16 to 30.

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Group	Statistic	df	Sig.	Statistic	df	Sig.
NumberOut_Installed	devsdsol	,317	30	,000	,778	30	,000
	arena	,537	30	,000	,275	30	,000
NumberOut_Repaired	devsdsol	,166	30	,034	,947	30	,142
	arena	,134	30	,178	,956	30	,241
NumberOut_NotRepaired	devsdsol	,162	30	,043	,955	30	,227
	arena	,134	30	,176	,956	30	,245
AvgTotalTime	devsdsol	,123	30	,200*	,962	30	,343
	arena	,088	30	,200*	,960	30	,317
AvgWaitingTime	devsdsol	,126	30	,200*	,958	30	,282
	arena	,133	30	,183	,967	30	,462
ServiceTechnician_Utilization	devsdsol	,113	30	,200*	,940	30	,091
	arena	,126	30	,200*	,976	30	,700
SupplyTechnician_Utilization	devsdsol	,106	30	,200*	,950	30	,167
	arena	,114	30	,200*	,962	30	,353

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure D.6: Normality test for the results for model-1.

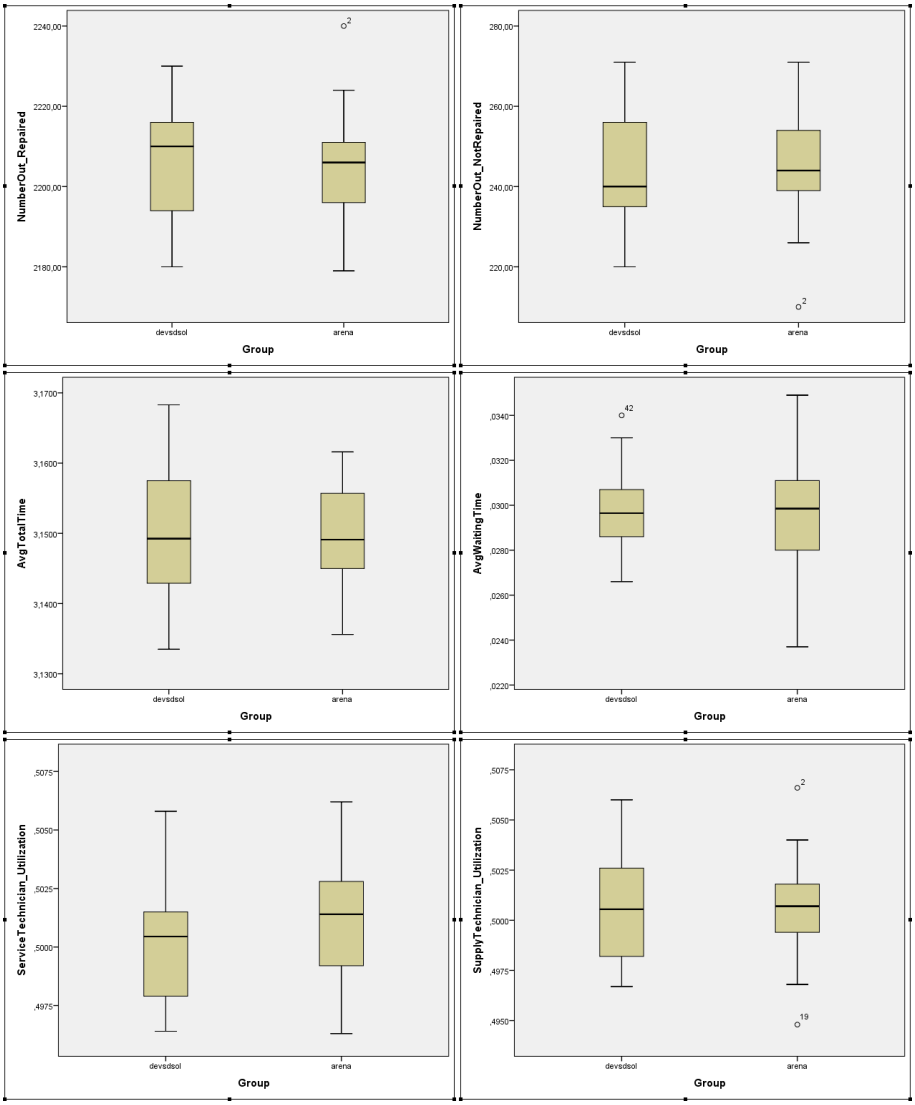


Figure D.7: Boxplots for the results for model-1.

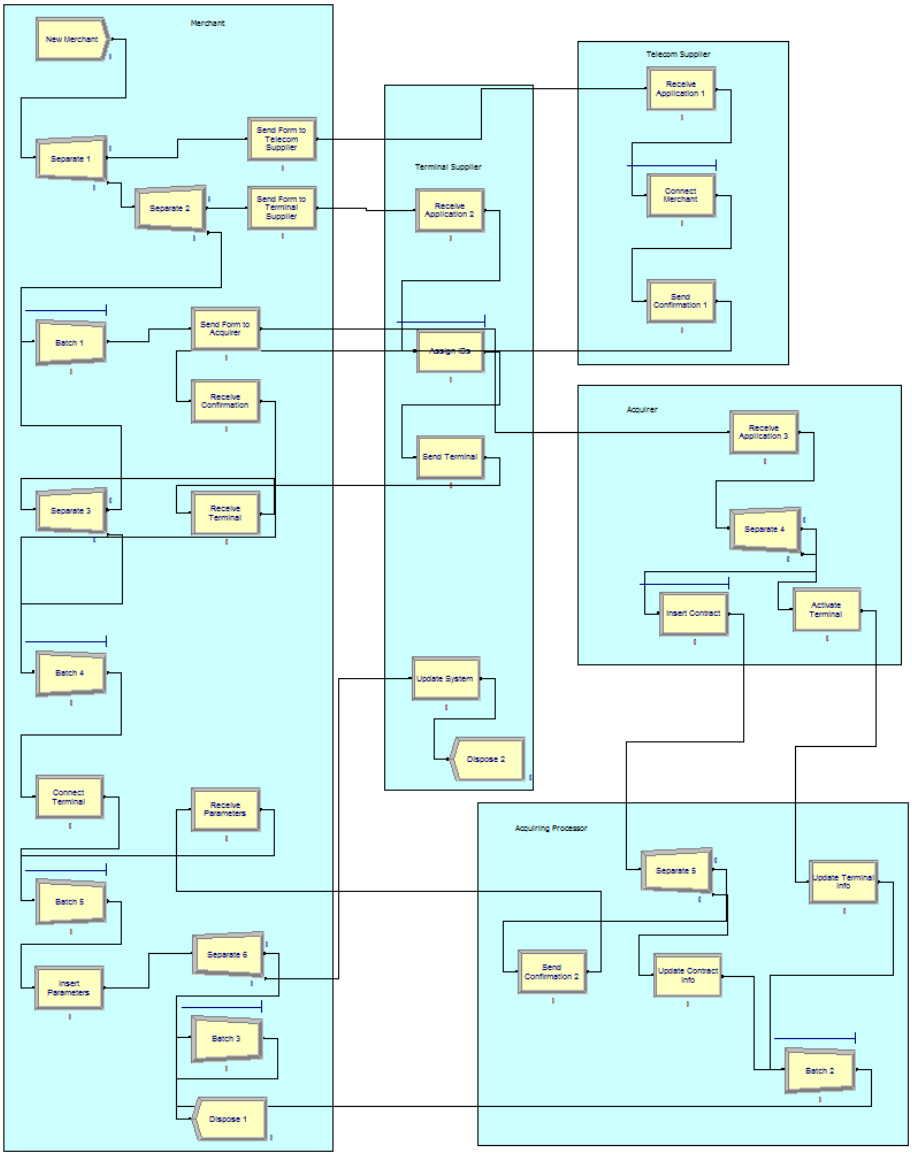


Figure D.8: Arena simulation model for the payment terminal application process model (model-2).

	Group	NumberOut	AvgTotalTime	AvgWaitingTime	Acquirer_Utilization	TelecomSupplier_Utilization	TerminalSupplier_Utilization
1	arena	907,00	5,1503	,3544	,9225	,9249	,9310
2	arena	908,00	5,1908	,3887	,9296	,9329	,9315
3	arena	907,00	5,1849	,3837	,9257	,9296	,9299
4	arena	906,00	5,1264	,3392	,9214	,9249	,9294
5	arena	907,00	5,1916	,3793	,9281	,9273	,9282
6	arena	908,00	5,0933	,2898	,9248	,9193	,9232
7	arena	907,00	5,1827	,3770	,9330	,9326	,9264
8	arena	907,00	5,2016	,3776	,9345	,9329	,9338
9	arena	907,00	5,2031	,4003	,9314	,9309	,9284
10	arena	907,00	5,2288	,4441	,9293	,9333	,9304
11	arena	907,00	5,2181	,4010	,9349	,9301	,9304
12	arena	906,00	5,2049	,3730	,9294	,9240	,9343
13	arena	906,00	5,2350	,4147	,9373	,9248	,9264
14	arena	907,00	5,2555	,4485	,9338	,9361	,9347
15	arena	907,00	5,1897	,3709	,9327	,9276	,9255

Figure D.9: Arena simulation results for model-2: Replications 1 to 15.

	Group	NumberOut	AvgTotalTime	AvgWaitingTime	Acquirer_Utilization	TelecomSupplier_Utilization	TerminalSupplier_Utilization
16	arena	907,00	5,1984	,4753	,9308	,9368	,9298
17	arena	907,00	5,1989	,4088	,9296	,9284	,9334
18	arena	907,00	5,2254	,4212	,9413	,9244	,9294
19	arena	908,00	5,2420	,4119	,9376	,9240	,9270
20	arena	907,00	5,1592	,3622	,9290	,9326	,9288
21	arena	906,00	5,1968	,3949	,9340	,9359	,9262
22	arena	907,00	5,2433	,4134	,9330	,9330	,9316
23	arena	907,00	5,2228	,4277	,9316	,9342	,9342
24	arena	907,00	5,2539	,4327	,9364	,9316	,9339
25	arena	906,00	5,1432	,3484	,9285	,9176	,9237

Figure D.10: Arena simulation results for model-2: Replications 16 to 25.

	Group	NumberOut	AvgTotalTime	AvgWaitingTime	Acquirer_Utilization	TelecomSupplier_Utilization	TerminalSupplier_Utilization
1	devsdsol	907,00	5,2286	,4755	,9308	,9362	,9308
2	devsdsol	907,00	5,2048	,4012	,9324	,9297	,9280
3	devsdsol	907,00	5,1806	,3723	,9274	,9285	,9316
4	devsdsol	907,00	5,1598	,3549	,9350	,9283	,9186
5	devsdsol	907,00	5,2296	,3974	,9344	,9226	,9347
6	devsdsol	907,00	5,2385	,4010	,9322	,9275	,9390
7	devsdsol	907,00	5,2168	,4133	,9334	,9308	,9315
8	devsdsol	907,00	5,2035	,3961	,9307	,9266	,9329
9	devsdsol	907,00	5,1631	,3562	,9323	,9293	,9240
10	devsdsol	908,00	5,1256	,3554	,9290	,9301	,9282
11	devsdsol	907,00	5,2014	,3697	,9291	,9221	,9355
12	devsdsol	907,00	5,2706	,4483	,9333	,9292	,9331
13	devsdsol	907,00	5,1825	,3686	,9316	,9311	,9260
14	devsdsol	907,00	5,1398	,3732	,9268	,9260	,9268
15	devsdsol	907,00	5,1517	,3474	,9307	,9246	,9260

Figure D.11: DEVSDSOL simulation results for model-2: Replications 1 to 15.

	Group	NumberOut	AvgTotalTime	AvgWaitingTime	Acquirer_Utilization	TelecomSupplier_Utilization	TerminalSupplier_Utilization
16	devsdsol	907,00	5,2260	,3961	,9360	,9274	,9256
17	devsdsol	907,00	5,1841	,3652	,9286	,9266	,9341
18	devsdsol	907,00	5,1813	,3601	,9341	,9347	,9288
19	devsdsol	907,00	5,1520	,3733	,9294	,9356	,9268
20	devsdsol	906,00	5,1916	,4348	,9271	,9346	,9302
21	devsdsol	908,00	5,1453	,3674	,9337	,9330	,9218
22	devsdsol	907,00	5,2451	,4203	,9346	,9276	,9356
23	devsdsol	907,00	5,1537	,3955	,9291	,9322	,9283
24	devsdsol	907,00	5,2102	,4243	,9309	,9361	,9316
25	devsdsol	907,00	5,2033	,4303	,9297	,9371	,9307

Figure D.12: DEVSDSOL simulation results for model-2: Replications 16 to 25.

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Group	Statistic	df	Sig.	Statistic	df	Sig.
NumberOut	devsdsol	,465	25	,000	,482	25	,000
	arena	,356	25	,000	,742	25	,000
AvgTotalTime	devsdsol	,099	25	,200*	,979	25	,866
	arena	,154	25	,130	,944	25	,184
AvgWaitingTime	devsdsol	,193	25	,018	,930	25	,086
	arena	,083	25	,200*	,984	25	,952
Acquirer_Utilization	devsdsol	,099	25	,200*	,968	25	,583
	arena	,095	25	,200*	,985	25	,963
TelecomSupplier_Utilization	devsdsol	,108	25	,200*	,964	25	,510
	arena	,146	25	,181	,945	25	,191
TerminalSupplier_Utilization	devsdsol	,075	25	,200*	,988	25	,987
	arena	,109	25	,200*	,959	25	,389

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure D.13: Normality test for the results for model-2.

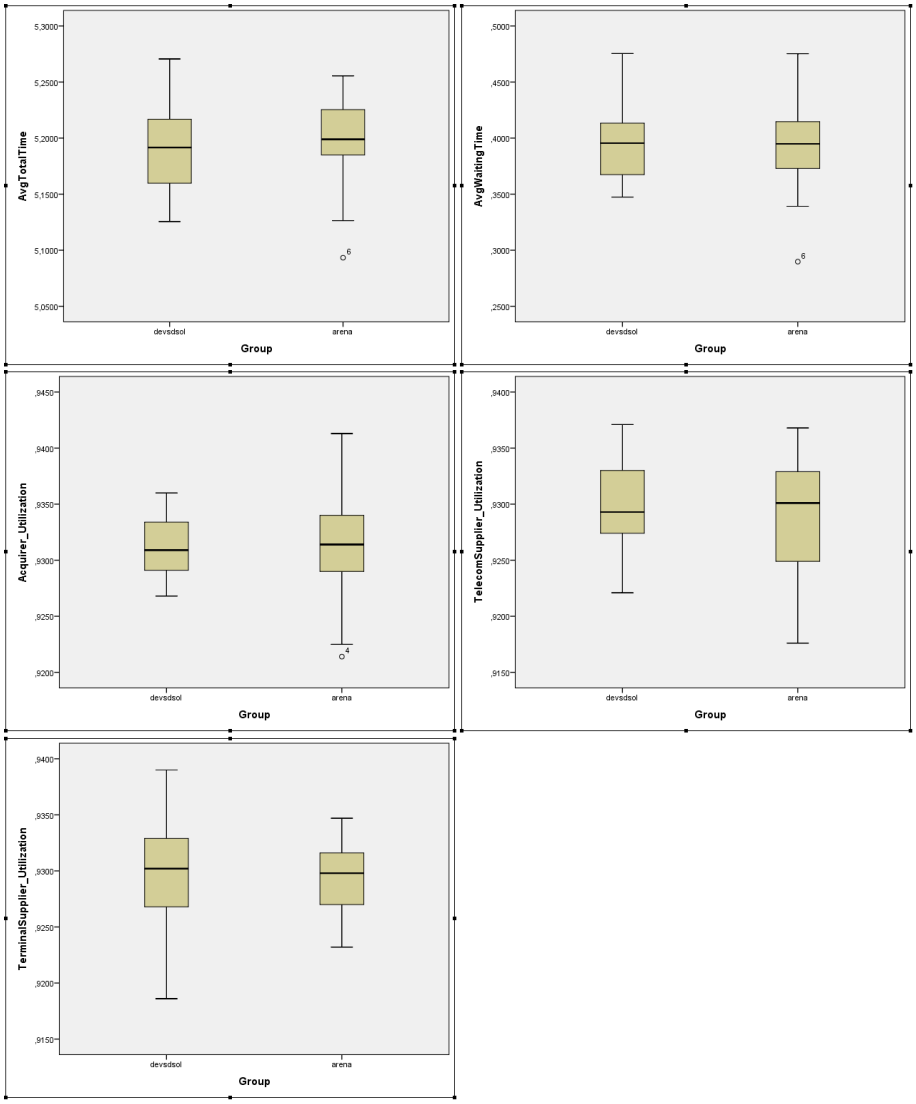


Figure D.14: Boxplot for the results for model-2.

Bibliography

[Acc13] Accenture, The accenture netherlands website, 2013, Last accessed July 1, 2013. Available at http://www.accenture.com/nl-en/Pages/index.aspx.

[AGY07] A. Achilleos, N. Georgalas, and K. Yang, An open source domain-specific tools framework to support model driven development of OSS, in *Proceedings of the 3rd European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '07)* (D. H. Akehurst, R. Vogel, and R. F. Paige, eds.), *Lecture Notes in Computer Science,* vol. 4530, Springer, 2007, pp. 1–16.

[Ack78] R. L. Ackoff, *The art of problem solving*, John Wiley & Sons, Inc., 1978.

[AE72] R. L. Ackoff and F. E. Emery, *On purposeful systems*, Aldine-Atherton, Inc., 1972.

[ACJG10] C. Agostinho, F. Correia, and R. Jardim-Goncalves, Interoperability of complex business networks by language independent information models, in *Proceedings of the 17th ISPE International Conference on Concurrent Engineering (New World Situation: New Directions in Concurrent Engineering)* (J. Pokojski, S. Fukuda, and J. Salwiński, eds.), Springer-Verlag London Limited, 2010, pp. 111–124.

[ASK04] A. Agrawal, G. Simon, and G. Karsai, Semantic translation of Simulink/stateflow models to hybrid automata using graph transformations, *Electronic Notes in Theoretical Computer Science*, vol. 109, 2004, pp. 43–56.

[AFR06] D. Amyot, H. Farah, and J.-F. Roy, Evaluation of development tools for domain-specific modeling languages, in *Proceedings of the 5th Workshop on System Analysis and Modeling: Language Profiles* (R. Gotzhein and R. Reed, eds.), *Lecture Notes in Computer Science,* vol. 4320, Springer, 2006, pp. 183–197.

[AB10] G. Arbez and L. G. Birta, The ABCmod conceptual modeling framework, in *Conceptual modeling for discrete event simulation* (S. Robinson, R. Brooks, K. Kotiadis, and D.-J. van der Zee, eds.), CRC Press, Taylor & Francis Group, 2010.

[AGRS13] P. Arcaini, A. Gargantini, E. Riccobene, and P. Scandurra, *Formal and practical aspects of domain-specific languages: Recent developments*, ch. 8, pp. 216–241, IGI Global, 2013.

[Are13] Arena, Arena simulation software, Rockwell Automation, Inc., 2013, Last accessed July 1, 2013. Available at http://www.arenasimulation.com.

[Ari01] L. B. Arief, *A framework for supporting automatic simulation generation from design*, Ph.D. thesis, Department of Computing Science, University of Newcastle, 2001.

[Ash57] W. R. Ashby, *An introduction to cybernetics*, Chapman & Hall Ltd., London, 1957.

[AK02] C. Atkinson and T. Kühne, Rearchitecting the UML infrastructure, *ACM Transactions on Modeling and Computer Simulation*, vol. 12(4), 2002, pp. 290–321.

[AK03] C. Atkinson and T. Kühne, Model-driven development: A metamodeling foundation, *IEEE Software*, vol. 20(5), 2003, pp. 36–41.

[BPL01] A. Bakshi, V. K. Prasanna, and A. Ledeczi, MILAN: A model based integrated simulation framework for design of embedded systems, in *Proceedings of the Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, ACM, 2001, pp. 82–93.

[BAO11] O. Balci, J. Arthur, and W. Ormsby, Achieving reusability and composability with a simulation conceptual model, *Journal of Simulation*, vol. 5, 2011, pp. 157–165.

[BO07] O. Balci and W. F. Ormsby, Conceptual modelling for designing large-scale simulations, *Journal of Simulation*, vol. 1(3), 2007, pp. 175–186.

[Bal01] O. Balci, A methodology for certification of modeling and simulation applications, *ACM Transactions on Modeling and Computer Simulation*, vol. 11(4), 2001, pp. 352–377.

[Bal12] O. Balci, A life cycle for modeling and simulation, *Simulation*, vol. 88(7), 2012, pp. 870–883.

[BBEN97] O. Balci, A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance, Developing a library of reusable model components by using the Visual Simulation Environment, in *Proceedings of the 1997 Summer Computer Simulation Conference*, 1997, pp. 253–258.

[BBEN98] O. Balci, A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance, Visual simulation environment, in *Proceedings of the 30^{th} Winter Simulation Conference (WSC '98)*, IEEE, 1998, pp. 279–288.

[Ban98] J. Banks, *Handbook of simulation: principles, methodology, advances, applications, and practice*, John Wiley & Sons, Inc., 1998.

[Bar98] R. R. Barton, Simulation metamodels, in *Proceedings of the 30th Winter Simulation Conference (WSC '98)*, IEEE, 1998, pp. 167–176.

[vB68] L. von Bertalanffy, *General system theory: foundations, development, applications, revised edition*, George Braziller, Inc., 1968.

[BBG05] S. Beydeda, M. Book, and V. Gruhn, *Model-driven software development*, Springer, 2005.

[BBG+06] J. Bézivin, F. Buttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow, Model transformations? Transformation models!, in *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS '06)* (O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds.), *Lecture Notes in Computer Science,* vol. 4199, Springer-Verlag Berlin Heidelberg, 2006, pp. 440–453.

[BDJ+03] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui, First experiments with the ATL model transformation language: Transforming XSLT into XQuery, in *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[BA07] L. G. Birta and G. Arbez, *Modelling and simulation - exploring dynamic system behaviour*, ch. 4, pp. 95–150, Springer-Verlag London Limited, 2007.

[BF06] B. S. Blanchard and W. J. Fabrycky, *Systems engineering and analysis, 4th edition*, Pearson Prentice Hall, 2006.

[BHT96] L. Blaxter, C. Hughes, and M. Tight, *How to research*, Open University Press, 1996.

[BAA08] C. A. Boer, B. A., and V. A., Distributed simulation in industry - a survey: Part 3 - the HLA standard in industry, in *Proceedings of the 40th Winter Simulation Conference (WSC '08)*, 2008, pp. 1094–1102.

[BVCH07] V. Bosilj-Vuksic, V. Ceric, and V. Hlupic, Criteria for the evaluation of business process simulation tools, *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 2, 2007, pp. 73–88.

[BD08] B. Bozlu and O. Demirörs, A conceptual modeling methodology: from conceptual model to design, in *Proceedings of the Summer Computer Simulation Conference*, Society for Modeling & Simulation International, 2008.

[vdBvDH+01] M. G. J. van den Brand, A. van Deursen, J. Heering, H. A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J. Vinju, E. Visser, and J. Visser, The ASF+SDF meta-environment: a

component-based language development environment, *Electronic Notes in Theoretical Computer Science*, vol. 44(2), 2001, pp. 5–10.

[BvET01] F. Brazier, P. van Eck, and J. Treur, Modelling a society of simple agents: From conceptual specification to experimentation, *Applied Intelligence*, vol. 14, 2001, pp. 161–178.

[BJT02] F. M. Brazier, C. M. Jonker, and J. Treur, Principles of component-based design of intelligent agents, *Data and Knowledge Engineering*, vol. 41, 2002, pp. 1–27.

[Bun79] M. Bunge, *Treatise on basic philosophy: Ontology II - a world of systems*, vol. 4, D. Reidel Publishing Company, Holland, 1979.

[BPSV03] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli, Modeling techniques in design-by refinement methodologies, in *System Specification and Design Languages*, Kluwer Academic Publishers, 2003, pp. 283–292.

[Bus00] A. Buss, Component-based simulation modeling, in *Proceedings of the 32nd Winter Simulation Conference (WSC '00)* (J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.), IEEE, 2000, pp. 964–971.

[ÇVS10] D. Çetinkaya, A. Verbraeck, and M. D. Seck, Applying a model driven approach to component based modeling and simulation, in *Proceedings of the 2010 Winter Simulation Conference* (B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.), IEEE, 2010, pp. 546–553.

[Çet13] D. Çetinkaya, MDD4MS project web page, Delft University of Technology, 2013, Last accessed June 21, 2013. Available at http://homepage.tudelft.nl/09s87/MDD4MS/.

[ÇMVS13] D. Çetinkaya, S. Mittal, A. Verbraeck, and M. D. Seck, Model-driven engineering and its application in modeling and simulation, in *Netcentric System of Systems Engineering with DEVS Unified Process, (S. Mittal and J.L. Risco-Martín, authors)*, CRC Press, Taylor & Francis Group, 2013, pp. 221–248.

[ÇV11] D. Çetinkaya and A. Verbraeck, Metamodeling and model transformations in modeling and simulation, in *Proceedings of the 2011 Winter Simulation Conference*, IEEE, 2011, pp. 3048–3058.

[ÇVS10a] D. Çetinkaya, A. Verbraeck, and M. D. Seck, A metamodel and a DEVS implementation for component based hierarchical simulation modeling, in *Proceedings of the 43rd Annual Simulation Symposium (ANSS '10), Part of the SpringSim '10*, Society for Modeling & Simulation International, 2010.

[ÇVS10b] D. Çetinkaya, A. Verbraeck, and M. D. Seck, Towards a component based conceptual modeling language for discrete event simulation, in *Proceedings of the 24th Annual European Simulation and Modelling Conference (ESM '10)*, EUROSIS-ETI, 2010, pp. 67–74.

[ÇVS11] D. Çetinkaya, A. Verbraeck, and M. D. Seck, MDD4MS: A model driven development framework for modeling and simulation, in *Proceedings of the Summer Computer Simulation Conference*, Society for Modeling & Simulation International, 2011, pp. 113–121.

[ÇVS12] D. Çetinkaya, A. Verbraeck, and M. D. Seck, Model transformation from BPMN to DEVS in the MDD4MS framework, in *Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium (TMS-DEVS '12)*, Society for Modeling & Simulation International, 2012, pp. 304–309.

[ÇVS13] D. Çetinkaya, A. Verbraeck, and M. D. Seck, BPMN to DEVS: Application of MDD4MS framework in discrete event simulation, in *Netcentric System of Systems Engineering with DEVS Unified Process, (S. Mittal and J.L. Risco-Martín, authors)*, CRC Press, Taylor & Francis Group, 2013, pp. 609–636.

[ÇVS14] D. Çetinkaya, A. Verbraeck, and M. D. Seck, Model continuity in discrete event simulation: A framework for model driven development of simulation models, *ACM Transactions on Modeling and Computer Simulation (submitted)*, 2014.

[CK12] C. C. Chang and H. J. Keisler, *Model theory*, *3rd edition*, Dover Publications, 2012.

[Che99] P. Checkland, *Systems thinking, systems practice: Includes a 30 year retrospective*, John Wiley & Sons Ltd., 1999.

[Cho02] N. Chomsky, *Syntactic structures*, Walter de Gruyter GmbH & Co., [1957] 2002.

[CJT10] L. B. Christensen, R. B. Johnson, and L. A. Turner, *Research methods, design, and analysis*, *11th edition*, Pearson Education, 2010.

[CBdMFS12] L. Chwif, J. Banks, J. de Moura Filho, and B. Santini, A framework for specifying a discrete-event simulation conceptual model, *Journal of Simulation*, vol. 7, 2012, pp. 50–60.

[Cre03] J. W. Creswell, *Research design: qualitative, quantitative, and mixed methods approaches*, *2nd edition*, Sage Publications, 2003.

[CH03] K. Czarnecki and S. Helsen, Classification of model transformation approaches, in *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[vDTVB09] C. E. van Daalen, W. A. H. Thissen, A. Verbraeck, and P. W. G. Bots, Methods for the modeling and analysis of alternatives, in *Handbook of Systems Engineering and Management* (A. P. Sage and W. B. Rouse, eds.), John Wiley & Sons, Inc., *2^{nd} edition*, 2009, pp. 1127–1169.

[Dal06] O. Dalle, OSA: an open component-based architecture for discrete-event simulation, in *Proceedings of the 20^{th} European Conference on Modelling and Simulation (ECMS '06)*, 2006, pp. 253–259.

[DGRMP10] A. D'Ambrogio, D. Gianni, J. L. Risco-Martín, and A. Pieroni, A MDA-based approach for the development of DEVS/SOA simulations, in *Proceedings of the Spring Simulation Multiconference*, Society for Modeling & Simulation International, 2010.

[DS99] T. Daum and R. G. Sargent, Scaling, hierarchical modeling, and reuse in an object-oriented modeling and simulation system, in *Proceedings of the 31^{st} Winter Simulation Conference (WSC '99)*, ACM, 1999, pp. 1470–1477.

[DBAS09] M. Dehayni, K. Barbar, A. Awada, and M. Smaili, Some model transformation approaches: a qualitative critical review, *Journal of Applied Sciences Research*, vol. 5(11), 2009, pp. 1957–1965.

[vDKV00] A. van Deursen, P. Klint, and J. Visser, Domain-specific languages: an annotated bibliography, *ACM SIGPLAN Notices*, vol. 35(6), 2000, pp. 26–36.

[Die06] J. L. G. Dietz, *Enterprise ontology: theory and methodology*, Springer, 2006.

[DN06] Y. Ding and N. Napier, Measurement framework for assessing risks in component-based software development, in *Proceedings of the 39^{th} Annual Hawaii International Conference on System Sciences (HICSS '06)*, IEEE, 2006.

[DdL09] J. N. Duarte and J. de Lara, ODiM: A model-driven approach to agent-based simulation, in *Proceedings of the 23^{rd} European Conference on Modelling and Simulation* (J. Otamendi, A. Bargiela, J. L. Montes, and L. M. D. Pedrera, eds.), 2009.

[Ecl06] Eclipse, Eclipse generative modeling technologies (GMT) project, The Eclipse Foundation, 2006, Last accessed June 21, 2013. Available at http://eclipse.org/gmt/.

[Ecl09] Eclipse, Eclipse modeling framework (EMF) project, The Eclipse Foundation, 2009, Last accessed June 21, 2013. Available at http://www.eclipse.org/modeling/emf/.

[EE08] H. Ehrig and C. Ermel, Semantical correctness and completeness of model transformations using graph and rule transformation, in *Proceedings of the 4th International Conference on Graph Transformations (ICGT '08)* (H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, eds.), *Lecture Notes in Computer Science,* vol. 5214, Springer, 2008, pp. 194–210.

[EEE+07] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer, Information preserving bidirectional model transformations, in *Proceedings of the Fundamental Approaches to Software Engineering*, *Lecture Notes in Computer Science,* vol. 4422, Springer-Verlag Berlin Heidelberg, 2007, pp. 72–86.

[ESB04] M. J. Emerson, J. Sztipanovits, and T. Bapty, A MOF-based metamodeling environment, *Journal of Universal Computer Science*, vol. 10, 2004, pp. 1357–1382.

[ESA05] ESA, *SMP 2.0 metamodel (issue 1 revision 2: EGOS-SIM-GEN-TN-0100)*, Technical report, ESA (European Space Agency), 2005.

[Far98] I. Farah, Approximate homomorphisms, *Combinatorica*, vol. 18(3), 1998, pp. 335–348.

[Fav04] J.-M. Favre, Towards a basic theory to model model driven engineering, in *Proceedings of the International Workshop on Software Model Engineering (WISME '04)*, 2004.

[FRR09] F. Fieber, N. Regnat, and B. Rumpe, Assessing usability of model driven development in industrial projects, in *Proceedings of the 4th European Workshop on From code centric to model centric software engineering: Practices, Implications and ROI* (T. Bailey, R. Vogel, and J. Mansell, eds.), 2009.

[Fis95] P. A. Fishwick, *Simulation model design and execution: Building digital worlds*, Prentice Hall, 1995.

[FB05] F. Fondement and T. Baar, Making metamodels aware of concrete syntax, in *Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)* (A. Hartman and D. Kreische, eds.), *Lecture Notes in Computer Science,* vol. 3748, Springer-Verlag Berlin Heidelberg, 2005, pp. 190–204.

[Fuj00] R. M. Fujimoto, *Parallel and distributed simulation systems*, John Wiley & Sons, Inc., 2000.

[FSV10] M. Fumarola, M. D. Seck, and A. Verbraeck, A DEVS component library for simulation-based design of automated container terminals, in *Proceedings of the 3rd International Conference on Simulation Tools and Techniques (SIMUTools '10)*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[GT10] J. Garcia and A. Tolk, Adding executable context to executable architectures: Shifting towards a knowledge-based validation paradigm for system-of-systems architectures, in *Proceedings of the 2010 Summer Computer Simulation Conference*, Society for Modeling & Simulation International, 2010, pp. 593–600.

[GPR13] A. F. Garro, F. Parisi, and W. Russo, A process based on the model-driven architecture to enable the definition of platform-independent simulation models, in *Revised selected papers from the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH '11)* (N. Pina, J. Kacprzyk, and J. Filipe, eds.), *Advances in Intelligent Systems and Computing,* vol. 197, Springer-Verlag Berlin Heidelberg, 2013, pp. 113–129.

[GW04] A. Gemino and Y. Wand, A framework for empirical evaluation of conceptual modeling techniques, *Requirements Engineering, Springer*, vol. 9, 2004, pp. 248–260.

[vG91] J. P. van Gigch, *System design modeling and metamodeling*, Plenum Press, 1991.

[GMB94] S. Greenspan, J. Mylopoulos, and A. Borgida, On formal requirements modeling languages: RML revisited, in *Proceedings of the International Conference on Software Engineering*, 1994, pp. 135–147.

[GKV01] D. G. Gregg, U. R. Kulkarni, and A. S. Vinzé, Understanding the philosophical underpinnings of software engineering research in information systems, *Information Systems Frontiers*, vol. 3(2), 2001, pp. 169–183.

[GL94] E. G. Guba and Y. S. Lincoln, Competing paradigms in qualitative research, in *Handbook of qualitative research* (N. K. Denzin and Y. S. Lincoln, eds.), Sage Publications, 1994.

[GKMM06] E. Guiffard, D. Kadi, J.-P. Mochet, and R. Mauget, CAPSULE: Application of the MDA methodology to the simulation domain, in *Proceedings of the European Simulation Interoperability Workshop*, SISO (Simulation Interoperability Standards Organization), 2006, pp. 181–190.

[GW12] G. Guizzardi and G. Wagner, Tutorial: conceptual simulation modeling with Onto-UML, in *Proceedings of the 2012 Winter Simulation Conference* (C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.), 2012.

[HKGV10] Z. Hemel, L. C. L. Kats, D. M. Groenewegen, and E. Visser, Code generation by model transformation: a case study in transformation modularity, *Software and Systems Modeling*, vol. 9, 2010, pp. 375–402.

[HHK10] F. Hermann, M. Hülsbusch, and B. König, Specification and verification of model transformations, in *Electronic Communication of the European Association of Software Science and Technology*, *Graph and Model Transformation (GraMoT),* vol. 30, 2010.

[HU04] J. Himmelspach and A. M. Uhrmacher, A component-based simulation layer for JAMES, in *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS '04)*, IEEE, 2004, pp. 115–122.

[Hol97] I. Holloway, *Basic concepts for qualitative research*, Blackwell Science, 1997.

[HZ05] X. Hu and B. P. Zeigler, Model continuity in the design of dynamic distributed real-time systems, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 35(6), 2005, pp. 867–878.

[HSV10] Y. Huang, M. D. Seck, and A. Verbraeck, LIBROS-II: Railway modelling with DEVS, in *Proceedings of the 2010 Winter Simulation Conference* (B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.), IEEE, 2010, pp. 2076–2086.

[Hub08] P. Huber, *The model transformation language jungle - an evaluation and extension of existing approaches*, Master's thesis, Business Informatics Group, Technische Universität Wien, 2008.

[HGG03] J. Hugh G. Gauch, *Scientific method in practice*, Cambridge University Press, 2003.

[IMA04] T. Iba, Y. Matsuzawa, and N. Aoyama, From conceptual models to simulation models: Model driven development of agent-based simulations, in *Proceedings of the 9th Workshop on Economics and Heterogeneous Interacting Agents*, 2004.

[IBM13] IBM, SPSS software, 2013, Last accessed July 1, 2013. Available at http://www-01.ibm.com/software/analytics/spss/.

[IDE99] IDEF, Integrated definition methods, 1999, Last accessed June 12, 2013. Available at http://www.idef.com/.

[IEE03] IEEE, Recommended practice for high level architecture (HLA) federation development and execution process (FEDEP), 2003, IEEE.

[IEE10] IEEE, Recommended practice for distributed simulation engineering and execution process (DSEEP), 2010, IEEE.

[INC06] INCOSE, Systems engineering handbook - a guide for system life cycle processes and activities (version 3), 2006, International Council on Systems Engineering.

[IRI11] IRISA, Kermeta workbench, Institut de Recherche en Informatique et Systèmes Aléatoires, 2011, Last accessed July 14, 2012. Available at http://www.kermeta.org/.

[ISI97] ISIS, Model integrated computing (MIC), Vanderbilt University, 1997, Last accessed June 21, 2013. Available at http://www.isis.vanderbilt.edu/research/MIC.

[ISO05] ISO/IEC, 19502:2005, International standard: Information technology - meta object facility (MOF), 2005.

[ISO10] ISO/IEC/IEEE, 24765, international standard: Systems and software engineering - vocabulary, 2010.

[JS09] E. Jackson and J. Sztipanovits, Formalizing the structural semantics of domain-specific modeling languages, *Software and Systems Modeling*, vol. 8(1), 2009, pp. 451–478.

[JLV02] P. H. M. Jacobs, N. A. Lang, and A. Verbraeck, D-SOL: a distributed Java based discrete event simulation architecture, in *Proceedings of the 34^{th} Winter Simulation Conference (WSC '02)*, 2002, pp. 793–800.

[JVM05] P. H. M. Jacobs, A. Verbraeck, and J. B. P. Mulder, Flight scheduling at KLM, in *Proceedings of the 37^{th} Winter Simulation Conference (WSC '05)*, 2005, pp. 299–306.

[JWLBB02] R. S. Janka, L. M. Wills, and J. Lewis B. Baumstark, Virtual benchmarking and model continuity in prototyping embedded multiprocessor signal processing systems, *IEEE Transactions on Software Engineering*, vol. 28(9), 2002, pp. 832–846.

[JABK08] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, ATL: A model transformation tool, *Science of Computer Programming*, vol. 72(1-2), 2008, pp. 31–39.

[JB06] F. Jouault and J. Bézivin, KM3: A DSL for metamodel specification, in *Formal Methods for Open Object-Based Distributed Systems*, *Lecture Notes in Computer Science,* vol. 4037, Springer, 2006, pp. 171–185.

[KV07] E. M. Kanacilo and A. Verbraeck, Assessing tram schedules using a library of simulation components, in *Proceedings of the 39^{th} Winter Simulation Conference (WSC '07)*, IEEE, 2007, pp. 1878–1886.

[KJB+09] T. Kapteijns, S. Jansen, S. Brinkkemper, H. Houët, and R. Barendse, A comparative case study of model driven development vs traditional development: The tortoise or the hare, in *Proceedings of the 4^{th} European Workshop on From code centric to model centric software engineering: Practices, Implications and ROI*, 2009.

[KN00] S. Kasputis and H. C. Ng, Composable simulations, in *Proceedings of the 32nd Winter Simulation Conference (WSC '00)* (J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.), 2000.

[KCG+08] R. Kewley, J. Cook, N. Goerger, D. Henderson, and E. Teague, Federated simulations for systems of systems integration, in *Proceedings of the 40th Winter Simulation Conference (WSC '08)*, 2008, pp. 1121–1129.

[KWHL03] C.-H. Kim, R. H. Weston, A. Hodgson, and K.-H. Lee, The complementary use of IDEF and UML modelling approaches, *Computers in Industry*, vol. 50, 2003, pp. 35–56.

[Kle08] J. P. C. Kleijnen, *Design and analysis of simulation experiments*, *International series in operations research and management science*, Springer Science and Business Media, 2008.

[Kle09] J. P. Kleijnen, Kriging metamodeling in simulation: A review, *European Journal of Operational Research*, vol. 192, 2009, pp. 707–716.

[KWB03] A. Kleppe, J. Warmer, and W. Bast, *MDA explained - the model driven architecture: practice and promise*, Addison-Wesley, 2003.

[Kli69] G. J. Klir, *An approach to general systems theory*, Litton Educational Publishing, Inc., 1969.

[KLV+10] G. L. Kolfschoten, S. G. Lukosch, A. Verbraeck, E. Valentin, and G. J. de Vreede, Cognitive learning efficiency through the use of design patterns in teaching, *Computers & Education*, vol. 54(3), 2010, pp. 652–660.

[KS03] A. Kossiakoff and W. N. Sweet, *Systems engineering: principles and practice*, *Wiley Series in Systems Engineering and Management*, John Wiley & Sons, Inc., 2003.

[KR08] K. Kotiadis and S. Robinson, Conceptual modelling: knowledge acquisition and model abstraction, in *Proceedings of the 40th Winter Simulation Conference (WSC '08)* (S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, eds.), IEEE, 2008, pp. 951–958.

[Küh06] T. Kühne, Matters of (meta-) modeling, *Software and Systems Modeling*, vol. 5(4), 2006, pp. 369–385.

[KWW11a] M. Kunze, M. Weidlich, and M. Weske, Behavioral similarity - a proper metric, in *Proceedings of the 9th International Conference on Business Process Management*, 2011, pp. 166–181.

[KWW11b] M. Kunze, M. Weidlich, and M. Weske, m^3 - a behavioral similarity metric for business processes, in *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS '11)*, 2011.

[LT08] J. Lachs and R. B. Talisse, *American philosophy: an encyclopedia*, Routledge, Taylor & Francis Group, 2008.

[dLV02] J. de Lara and H. Vangheluwe, AToM3: A tool for multi-formalism and meta-modelling, in *Proceedings of the Fundamental Approaches to Software Engineering* (R.-D. Kutsche and H. Weber, eds.), *Lecture Notes in Computer Science,* vol. 2306, Springer, 2002, pp. 174–188.

[Law03] A. M. Law, How to conduct a successful simulation study, in *Proceedings of the 35^{th} Winter Simulation Conference (WSC '03)*, 2003.

[LK91] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*, *2^{nd} edition*, McGraw-Hill, Inc., 1991.

[LDNA03] A. Ledeczi, J. Davis, S. Neema, and A. Agrawal, Modeling methodology for integrated simulation of embedded systems, *ACM Transactions on Modeling and Computer Simulation*, vol. 13(1), 2003, pp. 82–103.

[LWQY09] Y. Lei, W. Weiping, L. Qun, and Z. Yifan, A transformation model from DEVS to SMP2 based on MDA, *Simulation Modelling Practice and Theory*, vol. 17, 2009, pp. 1690–1709.

[LKPV03] A. Levytskyy, E. J. H. Kerckhoffs, E. Posse, and H. Vangheluwe, Creating DEVS components with the metamodelling tool AToM3, in *Proceedings of the 15^{th} European Simulation Symposium* (A. Verbraeck and V. Hlupic, eds.), Society for Modeling & Simulation International, 2003.

[Lin12] P. Linz, *An introduction to formal languages and automata*, *5^{th} edition*, Jones and Bartlett Learning, LLC, 2012.

[LDT13] C. J. Lynch, S. Y. Diallo, and A. Tolk, Representing the ballistic missile defense system using agent-based modeling, in *Proceedings of the Military Modeling & Simulation Symposium (MMS'13)*, Society for Modeling & Simulation International, 2013.

[MRB12] B. Magableh, B. Rawashdeh, and S. Barret, A framework for evaluating model-driven architecture, *American Journal of Software Engineering and Applications*, vol. 1, 2012, pp. 10–22.

[MR09] M. W. Maier and E. Rechtin, *The art of systems architecting*, *3^{rd} edition*, CRC Press, Taylor & Francis Group, 2009.

[MKY06] D. Mandelin, D. Kimelman, and D. Yellin, A bayesian approach to diagram matching with application to architectural models, in *Proceedings of the 28^{th} International Conference on Software Engineering (ICSE '06)*, ACM, 2006, pp. 222–231.

[MA09] S. T. March and G. N. Allen, Challenges in requirements engineering: A research agenda for conceptual modeling, in *Design Requirements Workshop*, *Lecture Notes in Business Information Processing,* vol. 14, Springer-Verlag Berlin Heidelberg, 2009, pp. 157–165.

[MCM13] Y. Martinez, C. Cachero, and S. Melia, MDD vs. traditional software development: A practitioner's subjective perspective, *Information and Software Technology*, vol. 55, 2013, pp. 189–200.

[MG05] T. Mens and P. V. Gorp, A taxonomy of model transformation, in *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, *Electronic Notes in Theoretical Computer Science,* vol. 152, 2005, pp. 125–142.

[MV11] B. Meyers and H. Vangheluwe, A framework for evolution of modelling languages, *Science of Computer Programming*, vol. 76, 2011, pp. 1223–1246.

[Mic05] Microsoft, Software factories, 2005, Last accessed June 21, 2013. Available at http://msdn.microsoft.com/en-us/library/bb977473.aspx.

[MBSF04] J. A. Miller, G. T. Baramidze, A. P. Sheth, and P. A. Fishwick, Investigating ontologies for simulation modeling, in *Proceedings of the 37th Annual Simulation Symposium (ANSS '04)*, IEEE, 2004.

[MRM13] S. Mittal and J. L. Risco-Martín, DEVSML 2.0, in *Netcentric System of Systems Engineering with DEVS Unified Process*, CRC Press, Taylor & Francis Group, 2013.

[MRMZ07] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, DEVSML: automating DEVS execution over SOA towards transparent simulators, in *Proceedings of the Spring Simulation Multiconference (SpringSim '07)*, Society for Modeling & Simulation International, 2007, pp. 287–295.

[MZRM+08] S. Mittal, B. P. Zeigler, J. L. Risco-Martín, F. Sahin, and M. Jamshidi, Modeling and simulation for systems of systems engineering, in *System of Systems: Innovation for the 21st Century* (M. Jamshidi, ed.), John Wiley & Sons, Inc., 2008, pp. 101–149.

[MD08] P. Mohagheghi and V. Dehlen, Where is the proof? - a review of experiences from applying MDE in industry, in *Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA '08)* (I. Schieferdecker and A. Hartman, eds.), *Lecture Notes in Computer Science,* vol. 5095, Springer-Verlag Berlin Heidelberg, 2008, pp. 432–443.

[MGS+11] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen, and M. Fritzsche, Where does model-driven engineering help? experiences

from three industrial cases, *Software and Systems Modeling*, 2011, Online article, DOI 10.1007/s10270-011-0219-7.

[MNA06] F. Moradi, P. Nordvaller, and R. Ayani, Simulation model composition using BOMs, in *Proceedings of the 10^{th} IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT'06)*, 2006, pp. 242–252.

[MFF+08] P.-A. Muller, F. Fondement, F. Fleurey, M. Hassenforder, R. Schnekenburger, S. Gérard, and J.-M. Jézéquel, Model-driven analysis and synthesis of textual concrete syntax, *Software and Systems Modeling*, vol. 7(4), 2008, pp. 423–442.

[NV09] S. Nain and M. Y. Vardi, Trace semantics is fully abstract, in *Proceedings of the 24^{th} Annual IEEE Symposium on Logic In Computer Science (LICS '09)*, 2009.

[Nan81] R. E. Nance, The time and state relationships in simulation modeling, *Communications of the ACM*, vol. 24(4), 1981, pp. 173–179.

[Nan84] R. E. Nance, Model development revisited, in *Proceedings of the 1984 Winter Simulation Conference* (S. Sheppard, U. W. Pooch, and C. D. Pegden, eds.), IEEE, 1984, pp. 74–80.

[Nan94] R. E. Nance, The conical methodology and the evolution of simulation model development, *Annals of Operations Research*, vol. 53, 1994, pp. 1–45.

[NK08] A. Narayanan and G. Karsai, Towards verifying model transformations, *Electronic Notes in Theoretical Computer Science*, vol. 211, 2008, pp. 191–200.

[NAS07] NASA, Systems engineering handbook (NASA/SP-2007-6105), 2007, NASA (National Aeronautics and Space Administration).

[NAT12] NATO, *Conceptual modeling (CM) for military modeling and simulation (M&S) (TR-MSG-058)*, Technical report, NATO RTO (Research and technology organisation), 2012.

[OB11] E. F. Y. Ogston and F. M. T. Brazier, AgentScope: multi-agent systems development in focus, in *Proceedings of the 10^{th} International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, 2011.

[Oli07] A. Olive, *Conceptual modeling of information systems*, Springer, 2007.

[OMG99] OMG, UML specification version 1.3, 1999, Object Management Group. Available at http://www.omg.org/spec/UML/1.3/.

[OMG03] OMG, Model driven architecture (MDA) guide version 1.0.1, 2003, Object Management Group. Available at http://www.omg.org/mda/specs.htm.

[OMG06] OMG, Meta object facility (MOF) core specification version 2.0, 2006, Object Management Group. Available at http://www.omg.org/spec/MOF/2.0/.

[OMG11a] OMG, Business process model and notation (BPMN) version 2.0, 2011, Object Management Group. Available at http://www.omg.org/spec/BPMN/2.0/PDF/.

[OMG11b] OMG, Meta object facility (MOF) 2.0 Query/View/Transformation (QVT), 2011, Object Management Group. Available at http://www.omg.org/spec/QVT/.

[OMG11c] OMG, Unified modeling language (UML) infrastructure version 2.4.1, 2011, Object Management Group. Available at http://www.omg.org/spec/UML/2.4.1/.

[Ong10] S. Onggo, Methods for conceptual model representation, in *Conceptual modeling for discrete event simulation* (S. Robinson, R. Brooks, K. Kotiadis, and D.-J. van der Zee, eds.), CRC Press, Taylor & Francis Group, 2010, pp. 337–354.

[Ö07] T. I. Ören, The importance of a comprehensive and integrative view of modeling and simulation, in *Proceedings of the Summer Computer Simulation Conference*, Society for Modeling & Simulation International, 2007, pp. 996–1006.

[OB91] W. J. Orlikowski and J. J. Baroudi, Studying information technology in organizations: research approaches and assumptions, *Information Systems Research*, vol. 2(1), 1991, pp. 1–28.

[OPB04] N. Oses, M. Pidd, and R. J. Brooks, Critical issues in the development of component-based discrete simulation, *Simulation Modelling Practice and Theory*, vol. 12(7-8), 2004, pp. 495–514.

[ON04] C. M. Overstreet and R. E. Nance, Characterizations and relationships of world views, in *Proceedings of the 36th Winter Simulation Conference (WSC '04)*, 2004, pp. 279–287.

[Pac00] D. K. Pace, Ideas about simulation conceptual model development, *Johns Hopkins APL Technical Digest*, vol. 21(3), 2000, pp. 327–336.

[PTTT09] P. Parviainen, J. Takalo, S. Teppola, and M. Tihinen, *Model-driven development: Processes and practices*, Technical report, VTT Technical Research Centre of Finland, 2009.

[Pet81] J. L. Peterson, *Petri Net theory and the modelling of systems*, Prentice Hall, 1981.

[Pid02] M. Pidd, Reusing simulation components: simulation software and model reuse: a polemic, in *Proceedings of the 34th Winter Simulation Conference (WSC'02)*, 2002, pp. 772–775.

[PMI08] PMI, A guide to the project management body of knowledge (PMBOK guide), an American national standard ANSI/PMI 99-001-2008, 2008, Project Management Institute, Inc.

[Pto07] Ptolemy, Ptolemy project, University of California at Berkeley, 2007, Last accessed June 21, 2013. Available at http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm.

[RB04] B. Ravindran and A. G. Barto, Approximate homomorphisms: A framework for non-exact minimization in markov decision processes, in *Proceedings of the 5th International Conference on Knowledge Based Computer Science (KBCS)*, 2004.

[RRIG09] J. Recker, M. Rosemann, M. Indulska, and P. F. Green, Business process modeling- a comparative analysis, *Journal of the Association for Information Systems*, vol. 10(4), 2009, pp. 333–363.

[Rei02] S. W. Reichenthal, The simulation reference markup language (SRML): A foundation for representing BOMs and supporting reuse, in *Proceedings of the Fall Simulation Interoperability Workshop*, 2002.

[RPMD09] J. Ribault, F. Peix, J. Monteiro, and O. Dalle, OSA: an integration platform for component-based simulation, in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)*, 2009.

[RMdMZ09] J. L. Risco-Martín, J. M. de la Cruz, S. Mittal, and B. P. Zeigler, eUDEVS: Executable UML with DEVS theory of modeling and simulation, *Simulation*, vol. 85(11-12), 2009, pp. 750–777.

[RAD+83] N. Roberts, D. F. Andersen, R. M. Deal, M. S. Garet, and W. A. Shaffer, *Introduction to computer simulation: a system dynamics modeling approach*, Productivity Press, 1983.

[Rob05] D. A. Robertson, Agent-based models to manage the complex, in *Managing Organizational Complexity: Philosophy, Theory and Application (A volume in Managing the Complex)* (K. Richardson, ed.), Information Age Publishing, Inc., 2005, pp. 417–430.

[Rob08a] S. Robinson, Conceptual modelling for simulation part I: definition and requirements, *Journal of the Operational Research Society*, vol. 59(3), 2008, pp. 278–290.

[Rob08b] S. Robinson, Conceptual modelling for simulation part II: a framework for conceptual modelling, *Journal of the Operational Research Society*, vol. 59(3), 2008, pp. 291–304.

[Rob04] S. Robinson, *Simulation: The practice of model development and use*, John Wiley & Sons Ltd., 2004.

[Rob06] S. Robinson, Conceptual modeling for simulation: issues and research requirements, in *Proceedings of the 38th Winter Simulation Conference (WSC '06)* (L. F. Perrone, B. Lawson, J. Liu, and F. P. Wieland, eds.), 2006, pp. 792–800.

[RBKvdZ10] S. Robinson, R. Brooks, K. Kotiadis, and D.-J. van der Zee (eds.), *Conceptual modeling for discrete-event simulation*, CRC Press, Taylor & Francis Group, 2010.

[RU06] M. Röhl and A. M. Uhrmacher, Composing simulations from XML-specified model components, in *Proceedings of the 38th Winter Simulation Conference (WSC '06)* (L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.), IEEE, 2006, pp. 1083–1090.

[Rum98] B. Rumpe, A note on semantics (with an emphasis on UML), in *Proceedings of the 2nd ECOOP Workshop on Precise Behavioral Semantics* (H. Kilov and B. Rumpe, eds.), Technische Universität München, 1998.

[Rus11] I. J. Rust, *Business process simulation by management consultants? introduction of a new approach for business process modeling and simulation by management consultants*, Master's thesis, Faculty of Technology, Policy and Management, Delft University of Technology, 2011.

[RÇSW11] I. Rust, D. Çetinkaya, M. D. Seck, and I. Wenzler, Business process simulation for management consultants: a DEVS-based simplified business process modelling library, in *Proceedings of the 23rd European Modelling and Simulation Symposium*, 2011.

[SA00] A. P. Sage and J. J. E. Armstrong, *Introduction to systems engineering*, John Wiley & Sons, Inc., 2000.

[Sar10] R. G. Sargent, Verification and validation of simulation models, in *Proceedings of the 2010 Winter Simulation Conference*, 2010.

[SE09] H. S. Sarjoughian and V. Elamvazhuthi, CoSMoS: a visual environment for component-based modeling, experimental design, and simulation, in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)* (O. Dalle, G. A. Wainer, L. F. Perrone, and G. Stea, eds.), 2009.

[SM12] H. S. Sarjoughian and A. M. Markid, EMF-DEVS modeling, in *Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium (TMS-DEVS '12)*, Society for Modeling & Simulation International, 2012.

[SLT07] M. Saunders, P. Lewis, and A. Thornhill, *Research methods for business students, 4^{th} edition*, Prentice Hall, 2007.

[Sch06a] M. Scheidgen, Model patterns for model transformations in model driven development, in *Proceedings of the 4^{th} Workshop on Model-Based Development of Computer-Based Systems*, 2006.

[Sch06b] D. C. Schmidt, Model-driven engineering, *IEEE Computer*, 2006, pp. 25–31.

[SV09] M. D. Seck and A. Verbraeck, DEVS in DSOL: adding DEVS operational semantics to a generic event-scheduling simulation environment, in *Proceedings of the Summer Computer Simulation Conference*, Society for Modeling & Simulation International, 2009, pp. 261–266.

[SCT03] A. F. Seila, V. Ceric, and P. Tadikamalla, *Applied simulation modeling*, Brooks Cole Publishing, 2003.

[Sel03] B. Selic, The pragmatics of model-driven development, *IEEE Software*, vol. 20(5), 2003, pp. 19–25.

[Sel06] B. Selic, Model-driven development: Its essence and opportunities, in *Proceedings of the 9^{th} IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, IEEE, 2006.

[Sha75] R. E. Shannon, *Systems simulation: the art and science*, Prentice Hall, 1975.

[Sha98] R. E. Shannon, Introduction to the art and science of simulation, in *Proceedings of the 30^{th} Winter Simulation Conference (WSC '98)* (D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, eds.), 1998.

[SHM07] G. A. Silver, O. A.-H. Hassan, and J. A. Miller, From domain ontologies to modeling ontologies to executable simulation models, in *Proceedings of the 39^{th} Winter Simulation Conference (WSC '07)* (S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.), 2007, pp. 1108–1117.

[Sim62] H. A. Simon, The architecture of complexity, in *Proceedings of the American Philosophical Society*, 1962, pp. 467–482.

[Som07] I. Sommerville, *Software engineering, 8^{th} edition*, Pearson Education Limited, 2007.

[Sta08] B. C. Stahl, *Information systems: critical perspectives*, Routledge, Taylor & Francis Group, 2008.

[SVB+06] T. Stahl, M. Volter, J. Bettin, A. Haase, and S. Helsen, *Model driven software development*, John Wiley & Sons Ltd., 2006.

[SBV+09] J. W. Sun, J. Barjis, A. Verbraeck, M. Janssen, and J. Kort, Capturing complex business processes interdependencies using modeling and simulation in a multi-actor environment, in *Proceedings of the 5th International Workshop on Advances in Enterprise Engineering and EOMAS 2009* (A. Albani, J. Barjis, and J. Dietz, eds.), *Lecture Notes in Business Information Processing,* vol. 34, Springer-Verlag Berlin Heidelberg, 2009, pp. 16–27.

[TKV10] A. A. Tako, K. Kotiadis, and C. Vasilakis, A conceptual modelling framework for stakeholder participation in simulation studies, in *Proceedings of the Operational Research Society Simulation Workshop (SW '10)*, 2010.

[TB11] O. O. Tanrıöver and S. Bilgen, A framework for reviewing domain specific conceptual models, *Computer Standards & Interfaces*, vol. 33, 2011, pp. 448–464.

[TT98] A. Tashakkori and C. Teddlie, *Mixed methodology: combining qualitative and quantitative approaches*, SAGE Publications, 1998.

[TS08] Y. M. Teo and C. Szabo, CODES: an integrated approach to composable modeling and simulation, in *Proceedings of the 41st Annual Simulation Symposium (ANSS '08)*, IEEE, 2008, pp. 103–110.

[TACS02] M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos, A theory of contexts in information bases, *Information Systems*, vol. 27, 2002, pp. 151–191.

[TDPHZ13] A. Tolk, S. Y. Diallo, J. J. Padilla, and H. Herencia-Zapana, Reference modelling in support of M&S: foundations and applications, *Journal of Simulation*, vol. 7, 2013, pp. 69–82.

[Tol13] A. Tolk, Truth, trust, and turing - implications for modeling and simulation, in *Ontology, Epistemology, and Teleology for Modeling and Simulation: Philosophical Foundations for Intelligent M&S Applications* (A. Tolk, ed.), *Intelligent Systems Reference Library,* vol. 44, Springer-Verlag Berlin Heidelberg, 2013, pp. 1–26.

[TDT08] A. Tolk, S. Y. Diallo, and C. D. Turnitsa, Mathematical models towards self-organizing formal federation languages based on conceptual models of information exchange capabilities, in *Proceedings of the 40th Winter Simulation Conference (WSC '08)*, 2008, pp. 966–974.

[TM04] A. Tolk and J. A. Muguira, M&S within the model driven architecture, in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2004.

[TAO08] O. Topçu, M. Adak, and H. Oğuztüzün, A metamodel for federation architectures, *ACM Transactions on Modeling and Computer Simulation*, vol. 18(3), 2008, pp. 10:1–29.

[TTH11] L. Touraille, M. K. Traoré, and D. R. C. Hill, A model-driven software environment for modeling, simulation and analysis of complex systems, in *Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium (TMS-DEVS '11)*, Society for Modeling & Simulation International, 2011, pp. 229–237.

[TGS+05] G. Trombetti, A. Gokhale, D. C. Schmidt, J. Greenwald, J. Hatcliff, G. Jung, and G. Singh, An integrated model-driven development environment for composing and validating distributed real-time and embedded systems, in *Model Driven Software Development* (S. Beydeda, M. Book, and V. Gruhn, eds.), Springer, 2005.

[Val11] E. Valentin, *Effective simulation studies using domain specific simulation building blocks*, Ph.D. thesis, Technische Universiteit Delft, 2011.

[VdL02] H. Vangheluwe and J. de Lara, Meta-models are models too, in *Proceedings of the 34th Winter Simulation Conference (WSC'02)* (E. Yücesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes, eds.), IEEE, 2002, pp. 597 – 605.

[VdLM02] H. Vangheluwe, J. de Lara, and P. J. Mosterman, An introduction to multi-paradigm modelling and simulation, in *Proceedings of the AI, Simulation and Planning in High Autonomy Systems (AIS '02)* (F. Barros and N. Giambiasi, eds.), Society for Modeling & Simulation International, 2002, pp. 163–169.

[VD02] A. Verbraeck and A. N. W. Dahanayake (eds.), *Building blocks for effective telematics application development and evaluation*, Delft University of Technology, 2002.

[Ver04] A. Verbraeck, Component-based distributed simulations: the way forward?, in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '04)*, ACM, 2004, pp. 141–148.

[VV08] A. Verbraeck and E. Valentin, Design guidelines for simulation building blocks, in *Proceedings of the 40th Winter Simulation Conference (WSC '08)* (S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, eds.), 2008, pp. 923–932.

[Vig09] A. Vignaga, *Metrics for measuring ATL model transformations*, Technical Report TR/DCC-2009-6, Department of Computer Science, Universidad de Chile, 2009.

[Vit03] P. Vitharana, Risks and challenges of component-based software development, *Communications of the ACM*, vol. 46(8), 2003, pp. 67–72.

[Wai09] G. A. Wainer, *Discrete-event modeling and simulation: A practitioner's approach*, CRC Press, Taylor & Francis Group, 2009.

[Wym67] A. W. Wymore, *A mathematical theory of systems engineering: the elements*, John Wiley & Sons, Inc., 1967.

[Wym93] A. W. Wymore, *Model-based systems engineering*, CRC Press, Taylor & Francis Group, 1993.

[YO06] L. Yilmaz and T. I. Ören, Prospective issues in simulation model composability: basic concepts to advance theory, methodology, and technology, *The MSIAC's (Modeling and Simulation Information Analysis Center's) M&S Journal Online*, vol. 2, 2006, pp. 1–7.

[Yu06] C. H. Yu, *Philosophical foundations of quantitative research methodology*, University Press of America, 2006.

[vdZKT+10] D.-J. van der Zee, K. Kotiadis, A. A. Tako, M. Pidd, O. Balci, A. Tolk, and M. Elder, Panel discussion: education on conceptual modeling for simulation - challenging the art, in *Proceedings of the 2010 Winter Simulation Conference* (B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Ycesan, eds.), 2010.

[vdZvdV07] D.-J. van der Zee and J. G. A. J. van der Vorst, Guiding principles for conceptual model creation in manufacturing simulation, in *Proceedings of the 39^{th} Winter Simulation Conference (WSC '07)* (S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.), IEEE, 2007, pp. 776–784.

[ZPK00] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of modelling and simulation: integrating discrete event and continuous complex dynamic systems*, *2^{nd} edition*, Academic Press, 2000.

[ZSC04] M. Zhou, Y. J. Son, and Z. Chen, Knowledge representation for conceptual simulation modeling, in *Proceedings of the 36^{th} Winter Simulation Conference (WSC '04)* (R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.), 2004, pp. 450–458.

List of Abbreviations

ATL ATLAS Transformation Language

BPMN Business Process Modeling and Notation

CBS Component Based Simulation

CM Simulation Conceptual Model

DEVS Discrete Event System Specification

DSL Domain Specific Language

DSML Domain-Specific Modeling Language

EMF Eclipse Modeling Framework

EMOF Essential MOF

GEMS Generic Eclipse Modeling System

GME Generic Modeling Environment

GMT Generative Modeling Technologies

GReAT Graph Rewriting and Transformation

GUI Graphical User Interface

IDE Integrated Development Environment

M2M Model to Model transformation

M2T Model to Text transformation

M&S Modeling and Simulation

MDA Model Driven Architecture

MDD Model Driven Development

MDD4MS Model Driven Development for Modeling and Simulation

MDE Model Driven Engineering

MIC Model Integrated Computing

MOF Meta-Object Facility

PISM Platform Independent Simulation Model

PSSM Platform Specific Simulation Model

QVT Query/View/Transformation

SysML Systems Modeling Language

UML Unified Modeling Language

List of Symbols

Table D.3: Symbols used in the thesis.

Symbol	*Meaning*
Sets	
S	set of all source systems
C	set of all contexts
L	set of all formal languages
M	set of models
V	set of secondary views of a model
M'	set of metamodels
L'	set of all metamodeling languages
P	set of all model transformation patterns
r	a set of transformation rules
Instances	
s, s_i	system
c, c_i	context
l, l_i	language
m, m_i	model
v, v_1	secondary view of a model
mm, mm_i	metamodel
l', l'_i	metamodeling language
g	grammar
$\widehat{g}$	extended grammar of g
$l(g)$	language generated by grammar g
p, p_i	model transformation pattern
Relations	
μ	model-of relation
γ	conforms-to relation
τ	instance-of relation
θ	transformed-to function

Summary

Modeling and simulation is an effective method for analyzing and designing systems and it is of interest to scientists and engineers from all disciplines. Simulation is the process of conducting experiments with a model for a specific purpose such as analysis, problem solving, decision support, training, entertainment, testing, research or education.

Several methodologies have been proposed in the literature to guide modelers through various stages of M&S and to increase the probability of success in simulation studies. Each methodology suggests a body of methods, techniques, procedures, guidelines, patterns and/or tools as well as a number of required steps to develop and execute a simulation model.

Most of the well known modeling and simulation methodologies state the importance of conceptual modeling in simulation studies and they suggest the use of conceptual models during the simulation model development process. However, the transformation from a conceptual model to an executable simulation model is often not addressed. Besides, none of the existing modeling and simulation methodologies provides guidance for formal model transformations between the models at different abstraction levels.

As a result, conceptual models are often not used explicitly in the further steps of the simulation study and a big semantic gap exists between the different models of the simulation project. This gap causes a lack of model continuity in many cases. The lack of model continuity has a potential risk of increased design and development costs due to unnecessary iterations. Model continuity is obtained if the initial and intermediate models are effectively consumed in the later steps of a development process and the modeling relation is preserved.

From the software engineering perspective, a (computer) simulation model can be seen as a software application and an M&S study can be seen as a software engineering project, as a simulation model is an executable program written in a programming language. The programming language can be either a general purpose programming language (such as C++, Java, etc.) or a specialized simulation programming language (such as SIMSCRIPT, SIMAN, SIMULA, etc.). In both cases, an interpreter, which may include a compiler, executes the simulation

model. Thus, software engineering methodologies can be applied to M&S and existing tools and techniques can be utilized.

In order to address the identified issues in the M&S field, this research proposes the application of a model driven development approach throughout the whole set of M&S activities and it proposes a formal MDD framework for modeling and simulation, which is called the MDD4MS framework. MDD is a software engineering methodology that suggests the systematic use of models as the primary means of a development process. MDD introduces model transformations between the models at different abstraction levels and proposes the use of metamodels for specifying modeling languages. In MDD, models are transformed into other models in order to (semi)automatically generate the final (software) system. In this research, the effects of applying an MDD approach throughout the whole set of M&S activities is tested with the proposed framework and its proof of concept implementation.

The MDD4MS framework presents an integrated approach to bridge the gaps between different steps of a simulation study by using metamodeling and model transformations. It mainly addresses the conceptual modeling and the simulation model development stages in M&S lifecycle and it can be incorporated into the existing methodologies for increasing the productivity, maintainability and quality of an M&S study.

The practical examples with the MDD4MS framework showed that if model transformations are complete and correct then an MDD4MS process obtains model continuity. Besides, it has been shown that using metamodeling and DSLs within the MDD4MS framework improves the conceptual modeling stage. As a result, applying an MDD approach in simulation reduces the gap between the conceptual modeling and the simulation model development stages in M&S lifecycle.

Samenvatting (in Dutch)

Titel: Model-gestuurde ontwikkeling van simulatiemodellen

Subtitel: Het definiëren en transformeren van conceptuele modellen naar simulatiemodellen, middels metamodellen en modeltransformaties

Modelleren en simuleren (M&S) is een effectieve aanpak voor analyse en ontwerp van systemen. Dit is van belang voor wetenschappers en ingenieurs in alle disciplines. Simulatie is het proces van het uitvoeren van experimenten binnen een model, gericht op een specifiek doel zoals analyse, het oplossen van een probleem, beslissingsondersteuning, training, vermaak, testen, onderzoek of onderwijs.

De literatuur biedt een keur aan methodologieën om modelontwikkelaars te leiden door de verschillende stadia van M&S en daarmee de kans op succes in simulatiestudies te verhogen. Elke methodologie biedt een verzameling van methoden, technieken, procedures, richtlijnen, patronen en/of gereedschappen, evenals een aantal vereiste stappen voor het ontwikkelen en uitvoeren van een simulatiemodel.

Van de goed bekende methodologieën voor modelleren en simuleren benadrukken de meeste het belang van conceptuele modellen in simulatiestudies en raden het gebruik aan van dergelijke modellen tijdens de ontwikkeling van een simulatiemodel. Slechts heel weinig methodologieën noemen de stap van conceptueel model naar uitvoerbaar model. Bovendien geldt dat geen enkele van de bestaande methodologieën richtlijnen geeft voor formele modeltransformaties tussen modellen op verschillende abstractieniveaus.

Dit heeft tot gevolg dat conceptuele modellen vaak niet expliciet gebruikt worden bij de latere stappen in een simulatiestudie en er een grote begripsmatige kloof bestaat tussen de verschillende modellen in een simulatieproject. Deze kloof veroorzaakt in veel gevallen een gebrek aan continuïteit in modellen. Een dergelijk gebrek aan continuïteit leidt tot het risico van verhoogde ontwerp en ontwikkelkosten door onnodige herhalingen. Modelcontinuïteit wordt verkregen door initiële en tussenliggende modellen effectief toe te passen in de latere stappen zodat de samenhang tussen modellen behouden blijft.

Vanuit het oogpunt van programmatuurontwikkeling kan een simulatiemodel gezien worden als een softwareprogramma en een M&S studie kan gezien worden als een

programmatuur-ontwikkeingsproject, want een simulatiemodel is een uitvoerbaar programma in een programmeertaal. Deze taal kan zijn een algemeen bruikbare programmeertaal (zoals C++, Java, etc.), of een gespecialiseerde simulatietaal (zoals SIMSCRIPT, SIMAN, SIMULA, etc.). In beide geval draait het simulatiemodel in een interpreter, al dan na voorbewerking door een compiler. Bestaande methoden en gereedschappen uit de programmatuurkunde kunnen derhalve toegepast worden op M&S.

In dit onderzoek wordt een raamwerk voorgesteld voor M&S, met de naam MDD4MS en gebaseerd op formele modellen. De effecten van het toepassen van MDD (Engelse afkorting voor modelgestuurde ontwikkeling) door de hele reeks van M&S-activiteiten is getoetst in de eerste experimentele implemetatie van dit raamwerk. MDD introduceert modeltransformaties tussen de modellen op verschillende abstractieniveaus en beveelt het gebruik aan van metamodellen (modellen van modellen) voor het specificeren van modelleertalen. In MDD worden modellen getransformeerd naar andere modellen teneinde het uiteindelijke softwaresysteem (half)automatisch te generen.

Het MDD4MS-raamwerk biedt een samenhangende aanpak om de afstand te verkleinen tussen de verschillende stappen in een simulatiestudie, met name door gebruik van metamodellering en modeltransformaties. Het richt zich vooral op de fasen van conceptueel modelleren en van simulatiemodelontwikkeling in de M&S levenscyclus. Het kan opgenomen worden in de bestaande methodologieën ten einde productiviteit, onderhoudbaarheid en kwaliteit in M&S-studies te verhogen.

Praktijkvoorbeelden van toepassing van MDD4MS hebben laten zien dat, als modeltransformaties volledig en correct uitgevoerd worden, het MDD4MS-proces modelcontinuïteit oplevert. Bovendien is aangetoond dat het gebruik van metamodellering en DSL's (domein-specifieke talen) in het kader van MDD4MS, het conceptueel modelleren ondersteunt en verbetert.

Op grond hiervan trekken we de conclusie dat toepassing van MDD in simulatiestudies de kloof verkleint tussen de conceptuele modellering en de eigenlijke simulatiemodelontwikkeling, niet alleen theoretisch maar ook in de praktijk.

About the Author

Deniz (Küçükkeçeci) Çetinkaya was born in Turkey on November 26, 1980. She received her B.Sc. with honors in Computer Science and Engineering from the Hacettepe University, Turkey in 2002. She received her M.Sc. in Computer Engineering from the Middle East Technical University, Turkey in 2005. She is living in the Netherlands since 2005. In June 2009, she started her PhD study under the supervision of Prof.dr.ir. Alexander Verbraeck in the Systems Engineering Section of the Faculty of Technology, Policy and Management at Delft University of Technology. During her research she presented her work at several international conferences in Europe and the USA. She also published two book chapters and made several presentations. Her research focuses on model driven software engineering and component based simulation.

Model Driven Development of Simulation Models

Modeling and simulation (M&S) is an effective method for analyzing and designing systems and it is of interest to scientists and engineers from all disciplines. This thesis proposes the application of a model driven software development approach throughout the whole set of M&S activities and it proposes a formal model driven development framework for modeling and simulation, which is called MDD4MS.

The MDD4MS framework presents an integrated approach to bridge the gaps between different steps of a simulation study by using metamodeling and model transformations. The practical examples with the MDD4MS framework showed that the framework is applicable and useful in the business process modeling and simulation domain.

This thesis mainly addresses the conceptual modeling and the simulation model development stages in the M&S lifecycle and the proposed framework can be incorporated into existing simulation methodologies for increasing the productivity, maintainability and quality of M&S projects.

ISBN 978-94-6108-513-9

TU Delft Delft University of Technology