Automated Abstraction of Discrete-Event Simulation Models using State-Trace Data

Tekinay, C.

DOI
10.4233/uuid:e4d3be75-c184-4f24-a127-6d8af9b30550

Publication date
2022

Document Version
Final published version

Citation (APA)
Tekinay, C. (2022). *Automated Abstraction of Discrete-Event Simulation Models using State-Trace Data*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:e4d3be75-c184-4f24-a127-6d8af9b30550

AUTOMATED ABSTRACTION *of* DISCRETE-EVENT SIMULATION MODELS *using* STATE-TRACE DATA

ÇAĞRI TEKİNAY

Automated Abstraction of Discrete-Event Simulation Models using State-Trace Data

Çağrı Tekinay

Cover Suzan Doornwaard | www.elephantpath.net
Printing Ridderprint | www.ridderprint.nl
ISBN 978-94-6458-300-7

Automated Abstraction of Discrete-Event Simulation Models using State-Trace Data

Proefschrift

Ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus, Prof.dr.ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op vrijdag 17 juni 2022 om 10:00 uur
door

Çağrı TEKİNAY
Master of Science in Information Systems
Middle East Technical University, Ankara, Turkije
geboren te Malatya, Turkije

Dit proefschrijf is goedgekeurd door de promotor:
Prof. dr. ir. A. Verbraeck

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. A. Verbraeck	Technische Universiteit Delft, promotor

Onafhankelijke leden:

Prof. dr. M.E. Warnier	Technische Universiteit Delft
Prof. dr. ir. J.H. Kwakkel	Technische Universiteit Delft
Dr. Y. Huang	Technische Universiteit Delft
Prof. dr. H.L.M. Vangheluwe	Universiteit Antwerpen, België
Prof. dr. S.J.E Taylor	Brunel University London, UK
Dr. G. Yücel	Boğaziçi University, Turkije

Canım aileme…

Contents

List of Abbreviations

DESS	Differential Equation System Specification
DEVS	Discrete-Event System Specification
DSDEVS	Dynamic Structure Discrete-Event System Specification
DSOL	Distributed Simulation Object Library
DTSS	Discrete Time System Specification
EMMA	Episodes Mining using Memory Anchor
FEM	Frequent Episode Mining
FIFO	First In First Out
FIMA	Frequent Itemset Mining using Memory Anchor
GSPS	General System Problem Solver
GTU	Generalized Traffic Unit
I/O	Input/Output
KPI	Key Performance Indicator
LMRS	Lane change Model with Relaxation and Synchronization
M&S	Modeling and Simulation
maxwin	Maximum Window Bound
minsup	Minimum Support Threshold
MRM	Multiresolution Modeling
PBL	Project Bound List
RNG	Random Number Generator
SPM	Sequential Pattern Mining
TKE	Top-K Episode Mining
V&V	Validation and Verification

Chapter 1

Introduction

1 Introduction

As a consequence of increased globalization and the ever-growing demand for faster, more reliable, and cheaper services, the systems we encounter and interact with in our daily lives have grown tremendously in size and complexity. It is no longer an unthinkable scenario that while we are on our way to work in our self-driving cars, we manage to secure the best flight deal out of hundreds of available options, using the help of intelligent virtual assistants on our latest technology phones which were delivered from their factories to our doors within days after their public release. All of these actions, however, rely on the proper operations of large-scale complex systems.

Large-scale complex systems are characterized by a large number of interconnected components and their diverse interactions (Filip & Leiviskä, 2009; Šiljak, 1978). Typical examples of these systems include global supply chains, transportation and logistics networks, modern manufacturing systems, and power grids. As the demand for the development and optimization of large-scale systems is growing (Arthur et al., 1999), so is the need for better techniques to understand their underlying dynamic behavior, and to predict and manage their long-term performance. However, because of their scale and complexity, it is often too difficult and expensive to experiment with large-scale complex systems directly. Therefore, models are needed that can capture the complexity of these type of systems under study but are more practical to work with (Banks, 1998). A useful and powerful method for experimentation that can analyze and evaluate large-scale complex systems is *simulation* (Law, 2015; Shannon, 1975).

1.1 Simulation and Simulation Models

Simulation is the imitation of the operation of a real-world system or a conceptual system by means of executing and experimenting with a model of that system (Banks et al.,

2010). In computer simulation[1], the system of interest is first abstracted into a non-software specific *conceptual model*, which is subsequently transformed into a computer-executable *simulation model* (Robinson, 2008). Before experimenting with a simulation model, the suitability and the accuracy of a conceptual model built to represent a real-world system need to be *validated*, and the correctness of the transformation from that conceptual model to a computerized simulation model needs to be *verified*. Once the verification and validation are completed, the simulation model is subjected to a rigorous set of experiments to generate data for further analysis (Kleijnen, 2015).

A simulation model can be viewed as a collection of *objects* and their interrelations. Objects are characterized by one or more *attributes* and the *values* assigned to these attributes (Kiviat, 1967, 1969; Nance, 1981). The enumeration of all attribute values of an object at a particular instant represent the *state* of that object (Nance, 1981). Simulation models can be classified along three dimensions: *dynamic* (i.e., time-variant) vs. *static* (i.e., time-invariant), *discrete* (i.e., state changes happen instantaneously at discrete points in time) vs. *continuous* (state changes happen continuously with reference to time), and *deterministic* (i.e., non-random, non-probabilistic) vs. *stochastic* (i.e., random, probabilistic) simulation models (Banks & Carson, 1984; Law, 2015). In dynamic models where the behavior of a system over time (i.e., its dynamic behavior) is of interest, *time* is the indexing attribute[2] of an object or of the object's state. In such cases, the execution of the simulation model traces a conceptual history of the system's dynamic behavior in the form of time-ordered state changes (Nance, 1981). In discrete-event simulation models, the state of the model remains constant over intervals of time, and the values of the attributes only change at predetermined *event times*. An *event* represents a change in the object's state. In contrast, continuous simulation models allow the state of the model to change continuously over time. Such simulation models are considered *deterministic* if the output of a simulation model is always reproducible; that is, a known input or starting value fully predicts the

[1] The term "simulation" will be subsequently used in the dissertation to refer to "computer simulation".

[2] An indexing attribute is an attribute of an object that enables state transitions to be ordered.

sequence of state changes and the output (Banks et al., 2010). On the other hand, they are *stochastic* if random number generators are used to generate the output values and to determine the sequence of the state changes. In this dissertation, we are mainly focusing on dynamic, discrete-event simulation models with both deterministic and stochastic characteristics.

By using simulation models, system changes can be tested prior to their real-life implementation, without committing real resources or taking actual risks. For example, governments can evaluate the effects of potential new policies on different societal variables without disrupting ongoing operations; investors can analyze the performance and payoff characteristics of various market strategies without taking financial risks, and healthcare professionals can study and replicate clinical situations without risking lives. In some cases, simulation may be the only feasible tool for testing new strategies or designs as conducting experiments in the actual environment is near impossible (Shannon, 1976). Moreover, as simulations use an artificial clock to manage time, long-term scenarios can be run to replicate a real-life system's dynamic behavior in much shorter time spans (i.e., time compression), whereas scenarios that are naturally completed within milliseconds in wall-clock time can be slowed-down to simulation-seconds (i.e., time expansion). As such, simulation allows analysts to observe the behavioral transitions and study the chain of events leading to particular short and long-term phenomena. Especially this ability to mimic behavior over time distinguishes dynamic simulation models from (static) mathematical models. In the case of large-scale complex systems, simulation techniques are applied to study the dynamic behavior and improve the design of, among other things, modern manufacturing systems (Negahban & Smith, 2014), healthcare systems (Zhang, 2018), and energy systems (Keirstead et al., 2012; Negahban & Smith, 2014; Sola et al., 2020).

1.2 Large-scale Complex Simulation Models

One of the "grand challenges to tackle" (Nicol in Page et al., 1999, pp. 1509-1510) is the significant growth in the scale and complexity of simulation models over the years. The *scale of a simulation model* can be defined as the number of objects in the model and how much of the real world is represented, and the *complexity of a simulation model* can be defined

as the product of the number of states per object – also referred to as *resolution* – and the number of objects in the simulation model (Zeigler et al., 2000). With the increased capabilities of computer technology, we have been able to run simulation models that are larger in scale and higher in complexity (Davis & Bigelow, 1998; Zeigler et al., 2000). While these advances have allowed for more accurate representations of real-world systems, the ever-increasing scale and complexity of simulation models may eventually result in models that become too complex themselves to work with (Astrup et al., 2008; Chwif et al., 2000; Darema, 2004; Henriksen, 2008; Saysel & Barlas, 2006; Taylor et al., 2015) – giving rise to what we refer as *large-scale complex simulation models*. These models raise important new questions and challenges for the modeling and simulation (M&S) community (Arthur et al., 1999; Chwif et al., 2000; Robinson, 2001), including how models of such large scale and complexity can be expressed, modeled more efficiently, and validated, and what tools and techniques can be used for this. In what follows, we outline the main challenges of large-scale complex simulation models as well as existing attempts to address these challenges.

1.3 Large-scale Complex Simulation Models: Main Challenges

Following Nicol (Page et al., 1999, pp. 1509-1510), the main challenges of large-scale complex simulation models can be classified as the *problem of scale*, the *problem of complexity,* the *problem of performance vs. accuracy,* and the *problem of data.*

1.3.1 Problem of Scale

Because of their scale and complexity, large-scale complex simulation models require a lot of time and resources to develop, maintain, and optimize (Longo, 2011; Wieland & Pritchett, 2007). Increases in scale and complexity often result in an exponential growth in the amount of computational power required to execute a simulation model (Page et al., 1999). In the absence of sufficient computational power, large-scale complex simulation models suffer from extremely long execution times. A strategy that was developed to deal with the lengthy running times of these type of models is the use of parallel and distributed simulations (see Banks et al., 2010; Carothers et al., 2017; Fujimoto, 2015; Fujimoto, 2001; Nelson, 2016; Perumalla, 2006 for extended reviews and discussion of parallel and distributed simulations). These types of simulations distribute the execution

of a single simulation model over multiple processors (Fujimoto, 2001), resulting in a potentially significant speedup of the simulation execution compared to sequential and centralized execution of simulation models. When parallel and distributed simulation was introduced a few decades ago, the consensus was that it would provide a viable solution to the increasing demand for computational power necessary to execute computationally intensive large-scale complex simulation models (Sulistio et al., 2004). Although improved computational power and advanced technologies and techniques (Buyya et al., 2011; Fujimoto, 2015) to utilize such power indeed successfully address some of the challenges of running large-scale complex simulation models (Page et al., 1999), these solutions primarily offer a way to *run* these models more efficiently. However, they do not tackle the problem of how large-scale complex simulation models can be *modeled* more efficiently. In fact, there is consensus among scholars that increased computational capacity paradoxically forms one of the factors for the proliferation of large-scale complex simulation models (Chwif et al., 2000; Nelson, 2016; Page et al., 1999). Researchers like Nicol (Page et al., 1999, pp. 1509-1510) and Hester and Collins (2012, p. 410) argue that continuously increasing computing power available to us is "whetting our appetite" for larger and larger models. However, the amount of computation required to execute these larger models can be a bottleneck, as it increases vastly in relation to the scale of large-scale complex simulation models. More importantly, the techniques to better design, control, and interpret such models lag behind (Carothers et al., 2017; Fujimoto, 2016; Henriksen, 2008; Page et al., 1999; Tolk, 2012). Hence, addressing the challenges of running simulation models of large scale alone does not address the total set of issues for large-scale complex simulation models.

1.3.2 Problem of Complexity

All simulation models are abstractions of a real-world or a conceptual system, but some are more detailed than others (Davis & Tolk, 2007). When alternative implementations of simulation models of a system have the same scale, their complexity can be a useful metric to compare these models. Because the complexity of a simulation model depends on the product of model scale, the resolution of objects, and interactions among them (Davis & Bigelow, 1998; Zeigler et al., 2000). Therefore, the complexity of a simulation model will increase when one or more of these three aspects increases. Large-scale

complex simulation models typically have *high complexity* as a result of all of these aspects being inherently high. In this dissertation, we use the term *complex* to refer to *high-complexity*.

There are several reasons why simulation model complexity is increasing, resulting in the proliferation of large-scale complex simulation models. On the one hand, increasing complexity may be the result of a lack of experience of the modeler or simply having the possibility to include more. For instance, inexperienced modelers may feel insecure about what to include in their models and therefore end up including as much detail as possible. Moreover, as argued above, increasing available computing power removes constraints on the maximum model scale and complexity and feeds the tendency to model more. On the other hand, increasing complexity may be caused by interpretation difficulties, such as a lack of understanding of the underlying real-world system, the inability to make adequate abstractions of this system (i.e., conceptual model and simulation model), or even the lack of clear simulation objectives (Chwif et al., 2000). For instance, when the overall goal of the simulation study is unclear, modelers may have difficulty defining the scope of the model and end up including more and more detail that the end user may be interested in (Salt, 1993).

Large-scale complex simulation models raise new questions and bring along new challenges. In addition to the performance (i.e., computational cost of executing a simulation model) and resource (e.g., time, money, expertise) issues that were addressed earlier in the "problem of scale" subsection, these new challenges include a decrease in usability, reusability, modifiability, and extensibility of the models (Balci et al., 2017) and an increased difficulty in their validation and verification (V&V). Due to the underlying complexity of large-scale complex simulation models, the success of V&V activities relies on the use of multiple evaluation methods (Balci et al., 2000; Deslandres & Pierreval, 1991), as well as effective, affordable (in terms of resources and computational costs), and standardized software to tackle the increased number of tests, data processing work, and computational complexity of the models to be evaluated (Arthur et al., 1999; Balci, 1994, 1997; Birta & Özmizrak, 1996; Wang et al., 2019). However, such automation software and techniques and auxiliary *selection mechanisms* to pick and combine the correct set of methods are currently largely lacking (Roungas et al., 2018), making it challenging to

implement V&V principles for large-scale complex simulation models. This highlights the need for more research on methodologies for the validation and verification of large-scale complex simulation models.

Several strategies have been proposed to address the problem of complexity of large-scale complex simulation models. A strategy that deals with the complexity issues of large-scale complex simulation models is the use of *model abstraction*. Based on the concept of hierarchy originating from the field of systems theory (Anderson, 1972; Holland, 1996; 2000; Simon, 1991) and later adapted to the field of M&S, the idea of model abstraction is that complex models consisting of many and diverse interacting objects can be simplified by decomposing them into less detailed, coarse-grained sub-models using the hierarchical relationships among them (Fishwick, 1986; 1988; 1989; Zeigler et al., 2000). By applying various model abstraction techniques (see for taxonomies of model abstraction techniques; Frantz, 1995; Lee & Fishwick, 1996; Yilmaz & Ören, 2004), modelers are able to generate models that are executable, valid (i.e., principal assumptions of the original model are preserved but essential behavioral and structural elements are simplified) and of lower resolution compared to the original higher resolution model (Zeigler, 2019). In this way, the complexity of the original model can be reduced while its structure is maintained. With the use of model abstraction, multiple models at different levels of abstraction can be generated. Although the choice of an abstraction level is dependent on the goals and requirements of the simulation study (Fishwick, 1988), models with different levels of detail yield different types of insights about the same underlying system. In the context of large-scale complex simulation models, it is often impossible to fully capture the totality of the complex system in one model (Hofmann, 2004; Yilmaz & Ören, 2004). In this case, building multiple models with different levels of abstraction or resolution that collectively represent the underlying system allows for a better understanding of the behavior of this system. Multiresolution modeling (MRM) is an approach to build a collection of models at different levels of abstraction to represent the same system of interest (Davis & Bigelow, 1998; 2003; Davis & Tolk, 2007, Yilmaz et al., 2007). A simple MRM architecture is composed of a high-resolution model (i.e., more-detailed, fine-grained), a low-resolution model (i.e., less-detailed, coarse-grained), and a mapping logic that connects these two models (Petty et al., 2012). MRM enables modelers to

gradually develop low-resolution simulation models from high-resolution ones by establishing conceptually and analytically correct and consistent hierarchies among them. The importance of MRM in large-scale complex simulation models is well-recognized (Hofmann, 2004; Zeigler, 2019). However, MRM currently still requires modelers to design lower resolution models manually. Given their complexity, this is not a feasible task when dealing with large-scale complex simulation models (Yilmaz & Tolk, 2006). Therefore, automation methods to facilitate model abstraction might be of added value.

Another strategy that aims to deal with both performance and complexity issues of such models is the use of *metamodels* (Barton, 2015; Kleijnen, 1987). Providing "a model of a model" (Kleijnen, 1987), a metamodel replaces an expensive simulation model with another model that is generated by approximating the I/O function of the original one, e.g. as a set of linear equations with interaction effects. A metamodel treats the simulation model as a black box; that is, the simulation model's I/O is observed, and the parameters of the metamodel are estimated (Barton, 1992; Kleijnen, 1987). Metamodels are, therefore, simpler, computationally more efficient models than the original models (Kleijnen, 2015; Simpson et al., 2001) and can also be considered as one of the model abstraction techniques (Frantz, 1995). However, a fundamental limitation of this black box approach is that metamodels do not benefit from the time and state transition information that is present in the underlying simulation model and describes the dynamic behavior of the system (Nance, 1981). Therefore, metamodels cannot predict I/O relations well for set of inputs (interventions) that have not been used in estimating metamodel's parameters.

1.3.3 Problem of Performance vs. Accuracy

The problem of performance vs. accuracy emerges from both the problem of scale and the problem of complexity described above. The choice of resolution for a simulation model at a given scale can be understood as a trade-off between the level of accuracy of a simulation model and the computational cost of its execution (i.e., its performance). Understanding this trade-off requires insight into how the model scale and resolution influence accuracy and performance (Tekinay et al., 2012). For instance, a simulation model will perform better in terms of runtime when the number of simulation model objects decreases. However, a decrease in resolution may result in a decrease in model

accuracy (Zeigler et al., 2000). Thus, modelers are forced to make a decision between sacrificing the accuracy of the simulation model for better performance, potentially risking the validity of the simulation model consistent with the intended application of the model (Schlesinger et al., 1979), or increasing the computational capacity and the associated costs in order to improve the complex model's runtime.

In addition, Astrup et al. (2008) demonstrated that more complex models do not always result in better predictions. In their study on the simulation of forest growth, the authors compared the predictive ability of five simulation models with increasing complexity. Their results revealed that both the simplest and the most complex models had the poorest predictive ability, whereas the model characterized by intermediate complexity showed the best predictive ability. Hence, to achieve good predictive ability and performance, it is essential to develop methods to generate parsimonious simulation models.

1.3.4 Problem of Data

A final issue associated with large-scale complex simulation models is the problem of data. Over the years, the increase in the scale and complexity of simulation models has led to an associated growth in the volumes of data generated by these models. A type of data that is particularly affected by the increases in scale and complexity of simulation models is the trace data. Trace data, or state-sequence data, is of interest for several M&S purposes, including trace-driven input testing in the validation of dynamic simulation models (Balci, 1994) and process modeling (van der Aalst, 2011; 2016). More importantly, trace data allow modelers to observe the history of a (stochastic) simulation model run in order to gain insight in how complex phenomena evolve over time (Kemper & Tepper, 2007). However, when these trace data become huge in terms of *variety* (i.e., the number of different data sources or model components to be sampled) and length (i.e., the number of times for different sources or model components to be sampled), they confine modelers' ability to identify and utilize frequent behavioral patterns for model abstraction (Kim et al., 2017; Tolk, 2015). Data mining (Atluri et al., 2018; Gan et al., 2017) and machine learning methods (Pedrycz & Chen, 2014) have been designed to ease the process of discovering frequent patterns in temporal data and dimension reduction, that is, conversion of data from a high dimensional space into a low

dimensional space (Hinton & Salakhutdinov, 2006). Although such methods have proven to be useful for recognizing behavioral patterns within large volumes of trace data (Chapela-Campa et al., 2019; Lu et al., 2019; Song et al., 2009; van der Aalst, 2011), they have not yet been applied to automate dynamic simulation model abstraction. This would require techniques that can not only identify important behavioral patterns in state-trace data, but also generate lumped states at various abstraction levels to form models at a lower level of resolution. Combined with the previously addressed challenges of large-scale complex simulation models, this points to a missing link in existing methods of dealing efficiently with these type of simulation models.

1.4 Research Objective and Research Questions

As the scale and complexity of the systems that we interact with in our daily lives are growing tremendously, the issue of how to efficiently and effectively study these systems has been attracting more attention. Simulation models traditionally used to analyze and evaluate complex systems have given rise to new challenges in the form of large-scale complex simulation models. The previous sections have shown that existing modeling methods and techniques lack the mechanisms to deal with the problems of scale, complexity, accuracy vs. performance, and data that characterize large-scale complex simulation models. We posit that there is a need for a method that can help reduce the complexity of complex simulation models. In this research, we therefore aim to provide a method that automates the abstraction of large-scale complex simulation models using their state-trace data.

Research Objective: *To investigate to what extent the abstraction of large-scale complex discrete-event simulation models can be automated using their state-trace data.*

In order to achieve this objective, the following research questions will need to be answered:

Research Question 1: *To what extent do existing methods allow for the abstraction of large-scale discrete-event simulation models?*

In the modeling and simulation literature, several model abstraction techniques exist for large-scale and complex simulations. What are the shortcomings of conventional

methods for the abstraction of large-scale and complex simulations? What existing methods in other fields can be adapted to address these shortcomings?

Research Question 2: *How should state-trace data from large-scale complex discrete-event simulation models be prepared to be used for the automated abstraction method?*

What are the important characteristics of state-trace data that enable the abstraction of large-scale complex discrete-event simulation models? And would any preprocessing of the state-trace data be needed to help the automated abstraction process?

Research Question 3: *What considerations and actions are needed for the automated abstraction of the original model's dynamic behavior using state-trace data?*

What key characteristics of the original model are essential to be captured in the state-trace data to represent its dynamic behavior accurately?

Research Question 4: *How well does the abstracted model represent the behavior of the original large-scale complex simulation model?*

To evaluate the accuracy and the validity of the abstracted model, we will need to compare key statistics and performance measures obtained from the original model and the abstracted model using descriptive and inferential statistics. Because of our particular focus on model abstraction, the original model is treated as the system of which we generate state-trace data. Therefore, we evaluate whether the abstracted model can answer the same questions that the original system (i.e., the original model) could answer. This type of validity relation (i.e., model-system comparison) is similar to generating a data trace from a real-life system, evaluating the validity of the abstracted simulation model against this real-life system.

1.5 Research Methodology

The *research methodology* refers to the selection of the appropriate method(s) to conduct the research (Chen & Hirschheim, 2004). Various taxonomies have been proposed for the classification of research methodologies (Alavi et al., 1989; Cash & Nunamaker, 1989, 1990, 1991; Galliers, 1991; Orlikowski & Baroudi, 1991). Among these taxonomies, *quantitative* vs. *qualitative* (Cash & Nunamaker, 1989) stands out as one of the most consistent comparisons. The quantitative methods, such as formal methods for data analysis,

numerical methods like mathematical modeling, laboratory experiments are designed to be used in natural sciences to study natural phenomena. In contrast, qualitative methods like participant observations, interviews, among others, are more inclined towards social sciences wherein people with their social and cultural contexts form a fundamental part of the research (Howe, 2002). The research in this dissertation used simulation models to generate data and data mining, pattern recognition, and statistical methods were used to analyze the data. Therefore, the research is highly quantitative. Another categorization of research methods is the deductive reasoning vs. inductive reasoning (Markus, 2007). The underlying reasoning in this research is deductive, as the existing work in the literature has been studied, a hypothesis is proposed (in Section 1.4) and the hypothesis is tested to find support for the proposed hypothesis.

1.6 Research Instruments

The choice of *research instruments* to be used in research is dependent on the research objective, research questions, and research methodology (Creswell, 2009; Galliers, 1991). For this study, we will perform a *literature review* to obtain systematic knowledge on M&S, systems theory, and data mining in order to answer Research Question 1. To address Research Question 2, we will use insights from existing literature to present the DEVS formalism and its concepts of states and state-transitions, while *case studies* and *simulation experiments* will be conducted to demonstrate considerations for the use of state-trace data for the automated abstraction process. Research Question 3 will be informed by a literature review in the domain of data mining and M&S, and by *data analysis* and simulation experiments using the case studies. Specifically, data mining techniques will be presented, compared, and applied to the state-trace data to identify behavioral patterns at the desired level of abstraction. Finally, data analysis in the form of *descriptive and inferential statistics* will be used to address Research Question 4.

1.7 Thesis outline

In Chapter 2, we will provide an overview of the systems-theory-rooted foundation of M&S and introduce some of the key terms and concepts used in systems studies and the field of M&S. We will go over the theoretical frameworks that we use as the foundation of our research in greater detail, as well as conduct a review of previous efforts on multi-

resolution simulation models. We will look into the relevant studies that have been done in the data mining field, specifically in the area of temporal data mining. Following our review of the existing literature, we will describe the tasks and data types in the data mining field, focusing on temporal data mining, and present the rationale for combining temporal data mining and discrete-event modeling. Chapter 2 will therefore provide an answer to Research Question 1.

In Chapter 3, we will provide a detailed breakdown of our proposed method, emphasizing its strengths and introducing several theoretical and practical considerations in its application. The chapter will begin by formally describing the key concepts and algorithms used in our method. We then describe each step of the temporal data mining-based method to automate the abstraction of discrete-event simulation models and address considerations and actions for the modeler in the method's application. We will demonstrate and validate the proposed method's practical application in a case study of an exploratory M/M/1 queueing system throughout the chapter. Chapter 3 will provide a first answer to Research Questions 2, 3, and 4 since the method that will be presented in the chapter covers the characterization of the data, the automated model abstraction, and validation of the results.

Each step of the proposed method will be covered in Chapter 4 using a battlefield case from which an earlier version was introduced in (Tekinay et al., 2012), to demonstrate how to apply the method introduced in Chapter 3 and tailor it step-by-step to a relatively larger, more complex model with different model characteristics. First, we will provide a high-level description of the battlefield model and present the details of the particular scenario used in the case study. We will then present the conceptual model and describe the behavioral characteristics of the model components. We will then follow the same section structure in Chapter 3 and address all considerations and actions for the modeler. Finally, we will provide the validation process results and present our conclusions, which will extend the answers for Research Questions 2, 3 and 4.

Chapter 5 will apply our method to a microscopic traffic simulation model, which has a relatively larger number of model components and a larger state space than the battlefield model. We will begin the chapter with a high-level description of the traffic model before delving into the specific scenario used in the case study. We will then present the

conceptual model's details and describe the characteristics of the model entities and the road network. We will then address all considerations and actions for the modeler following the same section structure as in Chapters 3 and 4. Next, we will present model-specific adjustments to the considerations and explain certain adjustments made to apply the method to the traffic simulation case study, which will again extends the answers to Research Questions 2, 3 and 4. Consequently, we will finish the chapter by presenting our findings from the validation process and explaining our learnings from applying our method to a large-scale complex discrete-event simulation model.

Finally, Chapter 6 will conclude the dissertation by summarizing our findings by reflecting on the research objective and answering each research question based on our learnings from the case studies in Chapters 3, 4, and 5. Finally, we will finalize the chapter by discussing the relevance of our research and providing suggestions for future work.

Chapter 2

Background and Key Concepts

2 Background and Key Concepts

In the previous chapter, we outlined the research motivation, objective, and research questions to set the scope of the research in this dissertation. In this chapter, we present the fundamental concepts and existing work in systems studies and M&S relevant to this research. In addition, we discuss how the field of data mining can contribute to the study of large-scale complex simulation models. Many authors have elaborated on the strong relationship between systems theory and M&S (Ören, 1971; Ören & Zeigler, 2012; Praehofer, 1991; Wymore, 1967; Zeigler et al., 2000; Zeigler & Praehofer, 1989). After all, a simulation model is a representation and abstraction of a real-world system to study that system. In what follows, we first introduce some of the key concepts in systems theory, including system complexity, system hierarchy, and levels of system knowledge. We then proceed with a discussion of concepts from M&S, such as level of systems specification, morphism, homomorphism, model abstraction, and modeling formalisms. Finally, we will describe the tasks and the types of data in the data mining field, particularly in the area of temporal data mining, and present the rationale for combining temporal data mining and multiresolution discrete-event modeling as the foundation for our study.

2.1 Introduction to Systems and Systems Theory

A system can be viewed as an abstraction of (a portion of) the real world. In the systems theory literature, a system is typically defined as a set of interacting, interrelated *entities* or *parts* within an observable boundary (Ackoff & Emery, 1972; Boardman & Sauser, 2008; Hitchins, 2008; Klir, 2001; von Bertalanffy, 1968). These parts can be characterized by one or more *variables* or *attributes*, and the *values* assigned to these variables (Flood & Carson, 1993). The enumeration of the values for a set of variables is called *state* (Klir & Elias, 2012).

Systems theory further describes a system in terms of its *structure*, that is the inner constitution of that system, and its *behavior*, that is the outer manifestation of the inner transformations (Zeigler et al., 2000). The system structure, which corresponds to the *white-box system view* in systems theory, is the collection of the system state, state transition

mechanism, and state-to-output mapping. The external behavior, which corresponds to the *black-box system view* in systems theory, constitutes the output of the system based on its inputs (Skyttner, 2006). Knowing the structure of a system allows one to gain insight into the internal working or internal behavior of that system, as well as to deduce its external behavior (Zeigler et al., 2000). A more unifying definition of system concept by Wymore (1967) can be formally expressed by a 7-tuple, as shown in Definition 2.1.

Definition 2.1. Formal definition of a system (Wymore, 1967)

$$S = (T, X, \Omega, Q, \delta, Y, \lambda)$$

where

$T \subset R^{+}_{0,\infty}$	is the *time base*, a formalization of the independent variable time t, where $R^{+}_{0,\infty}$ is the positive reals including 0 and ∞
X	is the *set of all admissible input values*
$\Omega = \{\omega: T \rightarrow X\}$	is the *set of all admissible input segments*
Q	is the *set of state values*
$\delta = \Omega \times Q \rightarrow Q$	is the *transition function*: how the state changes when (various) inputs are fed into the system, or when the system is in a certain state
Y	is the *output set*: observable parameters
$\lambda = Q \rightarrow Y$	is the *output function*; mapping of the system state to the (resulting) output of the system

According to Gaines (1979), how a system is defined and distinguished from its environment is strongly influenced by the investigator's perspective and goals. Specifically, the investigator decides what parts and interactions among those parts need to be included within the boundary of the system he or she is interested in (Meadows, 2009; Weinberg, 1975). Each of these included parts can be formally described as a system in itself, and regarded as a subsystem within the system (Flood & Carson, 1993; Gaines, 1979).

2.1.1 Dynamic Systems

One of the main characteristics of the large-scale complex systems that are the focus of this research is that they are dynamic. A *dynamic system* is a system whose state changes over time, either at a stochastic or deterministic rate. Dynamic systems can be distinguished from *static* systems, whose state remains constant within the chosen observation frame of that system (Liu, 2015). The state of a dynamic system at a given time $t \geq 0$, as well as the information of what variables constitute that state at $t = 0$, is indicated by one or more *state variables*. The set of all possible values that those state variables can take over time is called *state space* (Vangheluwe, 2008).

Dynamic systems can be categorized based on their time and state representations (Wainer, 2009; Zeigler et al., 2000). A system is said to have a *continuous time base* (i.e., time evolves continuously) when a real number represents the time. In contrast, the system has a *discrete time base* (i.e., time advances in discrete portions) when an integer number represents the time. Similarly, state variables are described as continuous if the values of the state variables are from a continuous set represented as a real number, or discrete if the values of the state variables are represented as a finite set. As such, a system can be categorized into four classes (see Figure 2.1) based on the representation of its time base and state variables (Wainer, 2009; Zeigler et al., 2000):

(I) *Continuous variable dynamic systems:* Systems that have both a continuous time base and continuous state variable (Figure 2.1.a).

(II) *Discrete-time dynamic systems:* Systems that have a discrete time base but continuous state variables (Figure 2.1.b).

(III) *Discrete-event dynamic systems:* Systems that have a continuous time base but discrete state variables (Figure 2.1.c).

(IV) *Discrete dynamic systems:* Systems that have both a discrete time base and discrete state variable (Figure 2.1.d).

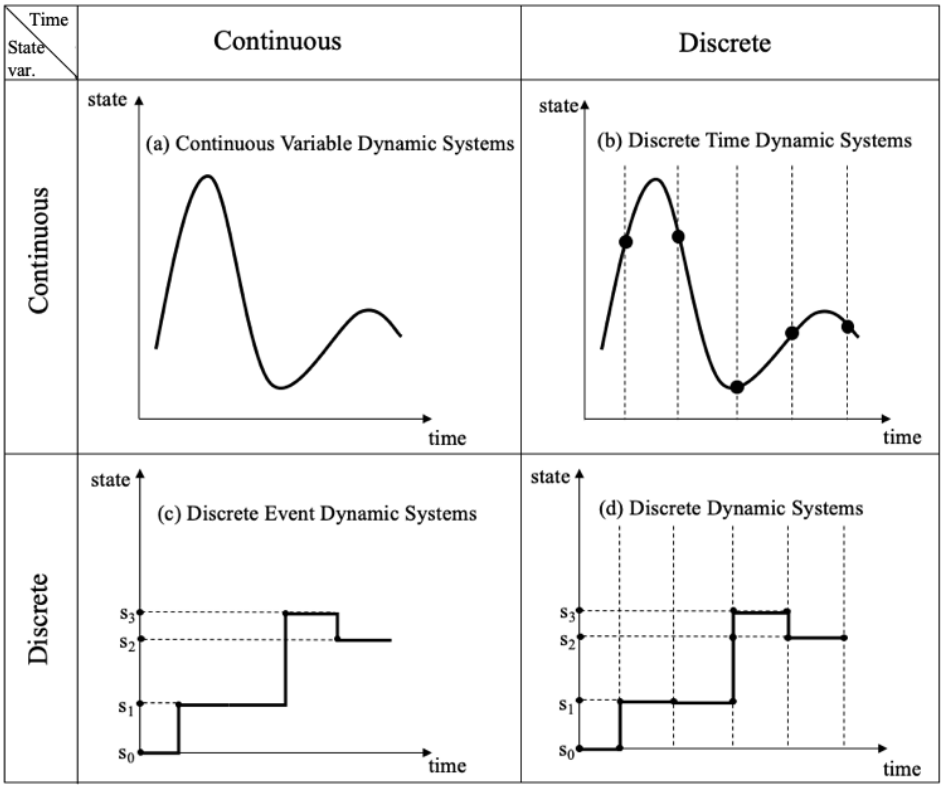

Figure 2.1. System classification based on the representation of time base/state variables (Zeigler et al., 2000)

2.1.2 Complexity and Hierarchy

Another key concept in systems theory and, in particular, the study of large-scale complex systems is complexity. Complex systems are generally characterized by many levels of ontological organization that can (not always) be divided or decomposed into smaller, less complex parts or subsystems (Anderson, 1972; Holland, 2000; Simon, 1962). The approach of systematically decomposing systems allows forming a hierarchy of systems specifications, thereby increasing the level of resolution of analysis and the system knowledge (Flood & Carson, 1993; Zeigler et al., 2000).

Typically, four types of hierarchy are distinguished in systems theory: order hierarchy, inclusion hierarchy, control hierarchy, and level hierarchy (Lane, 2006). *Order hierarchy* is

equivalent to the process of ordering a set of parts based on the values of their variables, for example, ordering sets based on their size or the number of elements in them. *Inclusion hierarchy* refers to the recursive relation among the parts of an organization; for example, the famous analogy of "Chinese boxes" (Simon, 1962) that is organized as a main box enclosing a second box within, and a third box is enclosed within the second box, and so on. The ontological claim of the inclusion hierarchy concept is that the container at level *m*, which is the main box in Simon's analogy, contains nothing but a certain number of other entities at level *m+1*, the entities at level *m+1* are only composed of entities at level *m+2,* and so on. According to Simon, the interactions among the parts that exist at the same hierarchical level in a complex system are *near-decomposable*, that is, a new decomposition of entities at a new spatial and temporal (spatio-temporal) level *m+1* is achievable from the system specification of level *m* (Simon, 1962). *Control hierarchy* refers to the ranking hierarchy within social organizations, for instance, the military ranks among soldiers in an army. The higher-ranking entities are entitled to command lower-ranking ones, and the lower-ranking entities are bound to obey the commands received from the higher-ranking entities (Lane, 2006). Finally, in *level hierarchy*, different parts in an ontological organization are postulated to exist at different spatio-temporal levels, and the higher-level parts at a particular spatio-temporal level may be either fully or partly composed of lower-level parts. In the case of the former, level hierarchy forms an inclusion hierarchy. In the case of the latter, some of the properties of lower-level entities and interactions change when forming a higher-level representation. Typical examples of level hierarchy can be found in cells, organs, individuals, and species.

2.1.3 Levels of Systems Knowledge

Epistemology, or the theory of knowledge, is the study of the origin and scope of knowledge and its justification. Forming a hierarchy of systems specifications through the process of systematical decomposition enables accessing system knowledge from distinct epistemological levels. The General System Problem Solver (GSPS; Klir, 1985) framework describes the systems knowledge that can be acquired from each epistemological level (see Table 2.1). Klir's taxonomy of systems uses notions like *investigator (or observer)*, *investigated object*, *environments* and *interactions* (between the investigator and the object) to describe each distinct levels of systems knowledge. In GSPS, the level of

knowledge accumulates when going up in the hierarchy as each level encapsulates all the knowledge available in all of the lower level systems.

Table 2.1. Hierarchy of epistemological levels of systems knowledge (Klir & Elias, 2012)

Level 4,5, …	Meta Systems	Relations between relations one level below
Level 3	Structure Systems	Relations between models one level below
Level 2	Generative Systems	Models that generate data one level below
Level 1	Data Systems	Observations and desired states one level below
Level 0	Source Systems	Empirical data source

Level 0, also called *source systems*, is the lowest level in the hierarchy of epistemological levels of systems. A source system is the source of empirical data and defined by a set of variables (basic or supporting) deemed relevant by the investigator, a set of potential states or values for those variables to obtain along their time-history, and their real-world interpretations. *Basic* variables of source systems can be partitioned into input and output variables, whereas the most common types of *supporting* variables are representing time, quantity (i.e., various populations of individuals of the same kind), and space. The set of aggregate states of all supporting variables forms a *support set* where changes in states of basic variables occur.

Level 1, or *data systems*, supplements the source system with data. Data is obtained by means of measurements or observation, or by the definition of *desirable states*, that is, the time-history of all basic variables within the support set.

Level 2 possesses knowledge that allows us to define one support-invariant characterization (e.g., time-invariant, space-invariant, etc.) of the relation among the basic variables for boundary conditions. This characterization describes a process with which the states of the basic variables (i.e., data) within a support set are generated. Therefore, this level is referred to as *generative systems*.

Level 3, or *structure systems*, is a representation of an *overall system* in terms of its subsystems that interact with each other in some way. An overall system is a system that represents

all of its associated lower level systems (source, data, and generative systems) based on the same support set.

Level 4 and higher, also referred to as *meta-systems*, define systems that consist of a set of systems defined at lower levels and a support-invariant meta-characterization. Meta-characterization is used to describe changes in system traits at lower level systems.

The GSPS framework allows defining a system as a part of the universe where the system and its observer coexist and interact for the purpose of dealing with fundamental system problems (Cellier, 1991). In the GSPS framework context, there are three fundamental system problems that involve moving between the levels of systems knowledge (Zeigler et al., 2000):

(I) *System analysis* is the effort to understand the behavioral characteristics of an existing or planned system. System analysis requires moving down the hierarchy, for instance a simulation which generates data under specific instructions fed by a model.

(II) *System inference* refers to the effort to infer system behavior through observation, or system structure from system behavior. System inference requires a transition from low-level system knowledge to a higher one, for instance from data to a probabilistic state machine.

(III) *System design* is the problem related to the ambition of finding a good design for a system that does not yet exist. Doing so requires moving up in the hierarchy to be able to generate data and then analyze such data.

The GSPS framework provides a hierarchy of epistemological levels of systems and identifies the system traits that are essentially participating in the system definition change (Skyttner, 2006). Although the framework is defined in the context of general systems theory, some of its key concepts, like systems knowledge and representation, are essential for simulation modeling (Zeigler et al., 2000).

2.2 Modeling and Simulation: Fundamental Concepts

The success of a simulation study is dependent on the understanding of fundamental entities and relationships in a modeling and simulation (M&S) process (Shannon, 1998;

Zeigler et al., 2000). These fundamental concepts (see Figure 2.2) defined by the M&S framework (Wainer, 2009; Zeigler et al., 2000) are as follows:

(I) *Source system:* The real or virtual environment in which we are interested in for modeling purposes. A source system is the source of *observable data* or *behavior database*, that is, time-indexed trajectories of variables gathered from observation or experimenting with the (source) system (Klir, 1985).

(II) *Experimental frame:* The specification of the conditions under which the system is observed or experimented with.

(III) *Model:* The set of instructions, equations, rules, or constraints for generating input/output (I/O) behavior.

(IV) *Simulator:* A computation system capable of executing a model to generate its behavior.

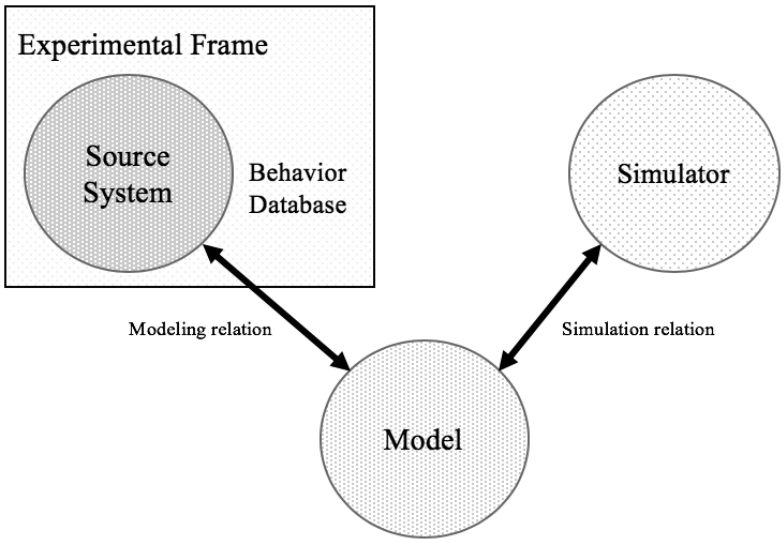

Figure 2.2. Basic entities in M&S and their relations (Zeigler et al., 2000)

In addition to these four basic entities in the M&S framework, there are two fundamental relationships between the basic entities:

(I) *Simulation relation, or model correctness,* is between a simulator and a model. The correctness of a simulator implies that the model's output trajectory is faithfully generated by the simulator given the models' initial state and the input trajectory.

(II) *Modeling relation*, or *validity*, refers to the relation between a model, a system and an experimental frame. Model validity is the degree to which a model properly represents its system counterpart under the conditions specified by the experimental frame of interest (Zeigler et al., 2000).

In this thesis, we study the validity relation between the original model and the abstracted model. As mentioned earlier in §1.4, we treat the original model as the system of which we generate state-trace data because of our particular focus on model abstraction. As a result, we investigate whether the abstracted model properly represents the original system (i.e., the original model) to evaluate its validity.

A simulation model is designed and specified to resemble its system counterpart in the epistemological hierarchy, in terms of parts, structure, relations, and input-output behavior, among others. The simulation model is a morphism of its system counterpart when the elements of the model and the system are properly placed into correspondence. Drawing a parallel between the levels system specifications and the levels of systems knowledge is important to understand and tackle systems problems, and subsequently, design and develop valid and coherent models. Such parallelism is presented by Zeigler et al. (2000) with a particular emphasis on the M&S context.

2.2.1 Levels of Systems Specification

Zeigler et al. (2000) formulated a hierarchy of systems specification levels that is similar to Klir's (1985) hierarchy of epistemological levels, but with more emphasis on the M&S context (see Table 2.2 for these levels and their correspondence to the levels of systems knowledge). The main difference between the two frameworks is that the systems specification hierarchy employs the concept of dynamic systems and acknowledges that simulation deals with the time-varying behavior of systems. Similarly, Zeigler's framework is committed to the use of concepts like input/output ports and modularity to explain the

levels of systems specification, whereas Klir's GSPS can include these terms but the framework is not dedicated to use them (Zeigler et al., 2000).

Table 2.2. Levels of systems knowledge and systems specification (Zeigler et al., 2000)

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

The *observation frame* corresponds to the source system at Level 0 in the systems knowledge hierarchy (Klir & Elias, 2012). It provides instructions on which variables are to be measured and how the behavior over time should be observed. A system in the system specification hierarchy interacts with other systems via its input and output ports; that is, it receives time-indexed input (input trajectories) through its input ports and generates time-indexed output (output trajectory) from its output ports.

The *I/O behavior* and *I/O function* correspond to the data system at Level 1. The collection of all time-stamped I/O pairs gathered by observation is called *I/O behavior* of a system. With the addition of the knowledge of an *initial state*, the I/O function indicates the functional relationship between the input and output; that is, the combination of an input trajectory and an initial state determines the *unique* output trajectory of a system.

The *state transition* of a system corresponds to the generative system at Level 2 in the systems knowledge hierarchy of Klir. This level provides instructions on how a state transition occurs in terms of input trajectory, current state, and the next (future) state so that the correct output trajectory is generated. A sequence of states (i.e., all future states $q_1, q_2, \ldots, q_n$ resulting from a given initial state q_0) that a system traces during its lifecycle ($q_0, q_1, q_2, \ldots, q_n$) is called a *state trajectory*.

The *coupled component* is the highest level in Zeigler's framework[3]. A component in a model corresponds to a part (or subsystem) in the system of interest. A coupled component is composed of components and their interactions coupled using ports; that is, the output port of a component is an input port of another component.

Additionally, Zeigler's System Specification Hierarchy describes model validity in relation to the levels of systems specification. The *experimental frame* is a critical entity in model validation. In essence, validation is the process of testing a model's validity by comparing its output trajectory to the system's output trajectory based on the input trajectories generated by the frame for both the source system and the model under test. For the model to be considered valid, both output trajectories that are fed back into the experimental frame must be similar within acceptable tolerance (Zeigler, 2019). Three types of validity are distinguished:

(I) The *replicative validity* is the most basic form of validation at the I/O relation level since it involves the comparison of observed data. Replicative validity of a model is affirmed if the behavior of the model, i.e., I/O behavior, and the system are a match within acceptable tolerance for all the experiments within the experimental frame.

(II) The *predictive validity* of a model is affirmed if the model has replicative validity and if it can generate (predict) future behavior given the past observations. Predictive validity requires agreement at I/O function level between the system and model.

(III) The *structural validity* is the strongest form of validity, and it requires agreement at state transition or at coupled component level. Structural validity requires the model to be capable of replicating the data observed from the system, and mimic the state transition of the system step-by-step and component-by-component.

---

[3] The system specification hierarchy (Zeigler et al., 2000) does not specify a level of systems specification matching Klir's meta-systems.

2.2.2 Morphism, Homomorphism and Model Abstraction

The systems specification hierarchy framework provides a stratification for constructing models that represent their system counterparts to the extent demanded by the intended use of the models (Zeigler et al., 2000). A systems specification in the context of M&S is the formal description of systems knowledge in the form of a (simulation) model specification. A simulation model is typically constructed at the generative systems or structure systems level, depending on the purpose of the simulation study. For instance, a more complex simulation model with coupled structures is constructed at level 3. The morphism relation between a simulation model and the original system implies that the elements of the model and the system are properly placed into correspondence (Klir, 2001; Zeigler et al., 2000).

A particular type of morphism relation between a pair of systems specifications at the state transition level is called *homomorphism*. A homomorphic relation (e.g., between a model and its original system[4]) indicates contingency upon a function from relevant entities of the original system *onto* the corresponding entities of the model system (i.e., the model) under which the relation among entities is preserved (Klir, 2001). Within this context, the *onto* function implies that the entity mapping (correspondence) is surjective; that is, the model is a simplification of the original system. In this case, the simplified model, or the *lumped model*, represents a part of a more complex *base model* with a certain degree of accuracy. Homomorphism suggests that there is a predefined mapping between the states of the base and lumped models, which is preserved under transitions; that is, for every base and corresponding lumped state, the respective next states to which they transit also correspond. It is assumed that the states of the lumped model *S'* and base model *S* are correspondent (congruent) but not identical, and simulating these corresponding states until their respective next states are encountered is done with different numbers of transitions (Zeigler, 2019). A specific case of this is illustrated in Figure 2.3, where a multi-step (micro) transition between the states B and F in the base model *S* is

[4] In M&S context, a model system can be the origin of another model system.

represented by one (macro) state transition between the states b and f in the lumped model *S'*.

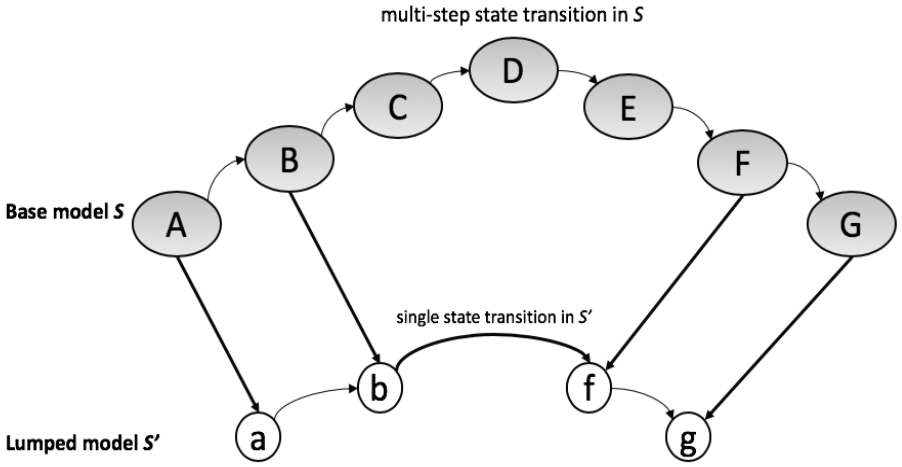

Figure 2.3. Homomorphism relation between the corresponding states and the state transitions of the base model S and the lumped model S'; based on Zeigler et al. (2000)

The notion of a homomorphism implies an error-free relation between the base/lumped model states, and achieving it ensures that the I/O behavior of the homomorphic models remains the same. That is, the values and the timings of the state transitions and the output trajectory of the base model within the same experimental frame are preserved in the lumped model. In Figure 2.3, this would imply that the total time delay of transitions B-C-D-E-F would be equal to the single time delay b-f. However, an entirely error-free model abstraction is not always possible in the real world (e.g., due to model complexity, stochasticity, time, and other resource limitations) and an error is introduced into the lumped model when an exact homomorphism is not achieved (Zeigler, 2019). A modeler may develop a lumped model that represents the entities in the base model quite accurately but not at a hundred percent. This type of morphism that has some error is called an *approximate morphism*. It should be noted that the concept of validity is not absolute; that is, a lumped model may still be valid if the error introduced by the approximate morphism is within an acceptable tolerance for goodness-of-fit.

2.2.3 Multi-level Morphic Model Pairs

As discussed earlier, it is often impossible to fully capture the totality of large-scale complex systems in one simulation model (Hofmann, 2004; Yilmaz & Ören, 2004). In such cases, multiresolution modeling (MRM) can be used to build a family of models with different levels of abstraction or resolution that collectively represent the underlying system (Davis & Bigelow, 1998; 2003; Davis & Tolk, 2007). Applying the concept of base/lumped model pairs to MRM, a lumped model is further simplified by constructing another lumped model of the original lumped model. In this newly formed morphic (base/lumped) model pair, the original lumped model is the base model. This approach to form multi-level morphic models by performing first order approximation in a recursive manner allows forming a hierarchy of models with varying state trajectories which, as a whole, provide a more complete description of a system than a single model description (Zeigler, 2019).

2.2.4 Basic Modeling Formalisms

A simulation model is designed and developed under certain constraints. A part of these constraints are imposed by the morphism relation between the system specification and the systems knowledge described in §2.2, and a part of them imposed by the non-functional requirements for the model; e.g., usability, reusability, modifiability, and extensibility (Balci et al., 2017). A systems specification needs to be expressed in a certain systems specification formalism – also referred to as a *modeling formalism*. A modeling formalism is a shorthand means of specifying a system, which implicitly sets constraints on the parts of the dynamic system (Zeigler et al., 2000). In other words, a modeling formalism consists of sets, relations on sets, and axioms on relations for expressing (simulation) models of dynamic systems. As discussed earlier in §2.2.1, dynamic systems are delineated with discrete or continuous time, and the state of the systems is specified using variables on a discrete or continuous domain (Wainer, 2009; Zeigler et al., 2000). Three basic modeling formalisms are developed to specify types of dynamic systems. The three formalisms are:

(I) *Discrete Time System Specification (DTSS)* represents systems over a discrete time base. It assumes a stepwise execution (Zeigler et al., 2000). At a particular

instant, the model is in a particular state, and it defines what the state at the next instant will be. If the state at time t is $q(t)$ and the input at time t is $x(t)$, then the state at time $t+1$ is $q(t+1) = \delta(q(t), x(t))$ where δ is the state transition function. Difference equations are an example of DTSS.

(II) *Differential Equation System Specification (DESS)* is a formalism that represents systems with continuous state over a continuous time base. DESS does not specify a next state directly through a state transition function. Instead, it specifies the rate of change of the state variables q_i through a derivative function f. Meaning that at any particular instant, given a state q and an input value x, the rate of change of the state can be obtained, i.e., $\frac{dq_i}{dx} = f(q_1(t), q_2(t), \ldots, q_n(t), x(t))$, $i = 1..n$, and can thus compute the state at any instant in the future using integration methods. Differential equations are an example of DESS.

(III) *Discrete Event System Specification (DEVS)* represents systems as piecewise constant state trajectories over a continuous time base. The state trajectories are produced by state transition functions δ_{int} and δ_{ext} that are activated by internal or external events.

The time base types for these three formalisms were addressed earlier in Figure 2.1.

2.2.4.1 DEVS Formalism

In this research, DEVS is chosen as the underlying modeling formalism. Our reasoning can be grouped into five categories:

(I) Large-scale complex systems can have components that are so diverse that they cannot all be expressed in a single formalism. Instead, multiple formalisms may be needed to model different system components; hence the concept of *multi-formalism modeling* (Vangheluwe & de Lara, 2002). In multi-formalism modeling, each system component can be modeled using the most suitable formalism; however, at the same time, a single formalism is identified into which each modeled component can be symbolically transformed (Vangheluwe & de Lara, 2002). The formalism space and existing behavior-preserving homomorphic relations between formalisms are shown in a formalism transformation graph;

see Figure 2.4**.** The DEVS formalism is the common denominator modeling formalism for the simulation of large-scale complex simulation models (Vangheluwe & de Lara, 2002; Vangheluwe, 2000).

(II) The DEVS formalism provides the building blocks for other modeling formalisms to be developed (e.g., Hierarchical DEVS (Zeigler et al., 2000), Fuzzy-DEVS (Kwon et al., 1996), Parallel-DEVS (Chow & Zeigler, 1994), Port-based DEVS[5] (Zeigler et al., 2000), Stochastic DEVS (Castro et al., 2010), Cell-DEVS (Wainer, 2002; Wainer & Giambiasi, 2001; 2002), and Markov DEVS (Seo et al., 2018)).

(III) The DEVS formalism supports hierarchical, modular and component-based construction of models (Wainer, 2009; Zeigler et al., 2000).

(IV) Conventional discrete event modeling approaches (e.g., activity cycle diagrams (Poole & Szymankiewicz, 1977), event graphs (Schruben, 1983), block diagrams (Schriber, 1989), process networks (Pritsker, 1979) mainly focus on the concepts of *activity, event* or *process*, and de-emphasize the concept of *state*. On the other hand, the DEVS formalism originates from the systems theory background and puts a particular emphasis on the notion of *state* (Praehofer & Pree, 1993).

(V) Different world views can be expressed as subclasses of DEVS:

a. the *process-interaction* world view describes the lifecycle of each entity through in the system,

b. the *activity scanning* world view describes the conditions that will trigger state changes (Vangheluwe, 2000; Zeigler et al., 2019), and

c. the *event-scheduling* world view describes the effect of each event on the state and on the future behavior of the system (typically the external

[5] Referred to as Classic DEVS with Ports (Zeigler et al., 2000, p. 84)

events are not part of this view). The case study models in Chapter 3 and 5 use the event-scheduling world-view. However, as discussed by Seck and Verbraeck (2009), the event-scheduling world view is harder to formulize than DEVS Formalism, which has a sound mathematical formalization for hierarchical models (see Definition 2.3 below). Furthermore, according to Vangheluwe and de Lara (2002), the underlying features of the event-scheduling worldview can be completely expressed using DEVS concepts (see Figure 2.4). Therefore, we will use the concepts from the DEVS formalism (e.g., time advance, memorlyless property) also to describe the model characteristics in Chapter 3 and 5 that have been implemented using an event-scheduling world view.

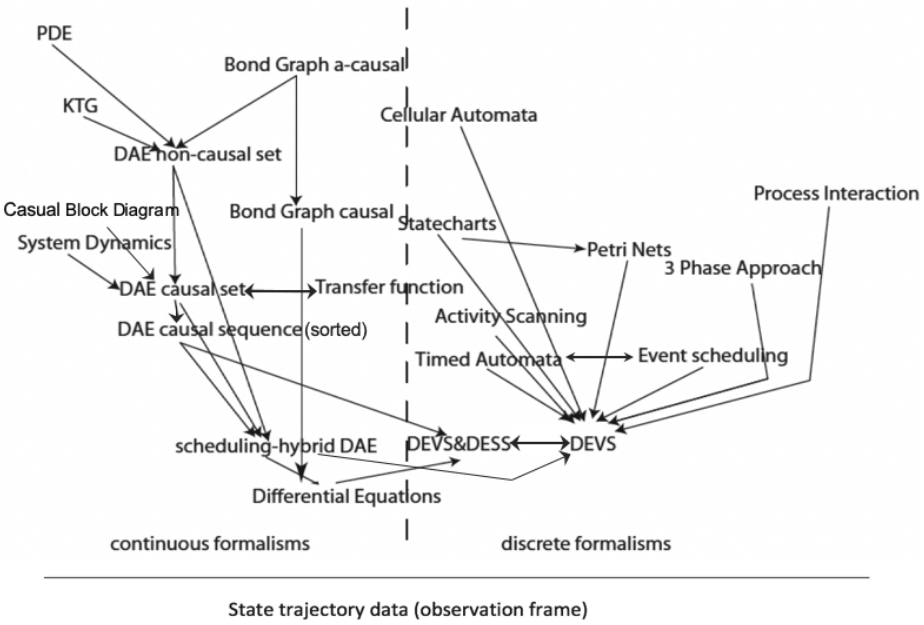

Figure 2.4. Formalism Transformation Graph (adjusted from Vangheluwe, 2008). The arrows indicate an existing homomorphic relationship between formalisms

DEVS is a general modeling formalism that allows the formal representation of systems whose I/O behavior is described by a sequence of events. The DEVS formalism allows describing systems as a composition of *atomic* and *coupled* components. Atomic models are expressed in the basic DEVS formalism, whereas coupled models are expressed in the coupled DEVS formalism containing components and coupling information (Zeigler et al., 2000).

An atomic DEVS model is a description of the autonomous behavior of a discrete event system represented as a sequence of deterministic transitions between sequential states over time (Vangheluwe, 2000). Additionally, it formally describes the mechanism to react to external inputs (events) and to generate output (events). A basic DEVS formalism for atomic DEVS models is a structure *M* as described in Definition 2.2.

Definition 2.2. Formal definition of an atomic DEVS model (Zeigler et al., 2000)

$$M = < X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda >$$

where

X	is the *set of input events*
Y	is the *set of output events*
S	is the *set of possible states*, where $s_0 \in S$ *is the initial state*
ta: $S \rightarrow \mathbb{T}^\infty$	is the *time advance function* that is used to determine the lifespan of a state. $\mathbb{T}$ is the time base, $\mathbb{T}^\infty = [0, \infty]$ is the set of non-negative real numbers plus infinity
$\delta_{ext}: Q \times X \rightarrow S$	is the *external state transition function*, where $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the total state set
	e is the elapse time since the last state transition
$\delta_{int}: S \rightarrow S$	is the *internal state transition function* that defines how a state of the system changes internally (when the elapsed time e reaches the lifetime of the state)
$\lambda: S \rightarrow Y$	is the *output function* that maps the current state on the output of the atomic model

This formalism is an extension of the system description of Wymore (1967) mentioned earlier in Definition 2.1, where the major changes are the addition of the *ta* function to make the duration of state changes more explicit, and the differentiation between δ_{int} and δ_{ext}. The basic DEVS formalism contains information about the state transition of a system as it corresponds to systems knowledge at the generative level and systems specification at state transition level (§2.1.3 and §2.2.1). For simplicity, the confluent state transition function $\delta_{con}(s, x)$ will be discussed after the basic state transition mechanism is described.

At any given time, a DEVS model is in some state, $s \in S$. Each possible state s has an associated time advance calculated by the time advance function $ta(s)$. In the absence of an external event, an atomic DEVS model will stay in a state s for a $ta(s)$ units of time. A state *transition* occurs by an external state transition function $\delta_{ext}(s,e,x)$ or an internal state transition function $\delta_{int}(s)$. When the calculated time advance expires, i.e., when the elapsed time $e = ta(s)$, the system outputs the value $\lambda(s)$, and transitions to a new state $\delta_{int}(s)$. This is called an *internal transition*. Alternatively, an external event $x \in X$ might occur before the consumption of the $ta(s)$, i.e., $e < ta(s)$. In that case, the model transitions to a new state given by the external state transition function $\delta_{ext}(s,e,x)$. This is called an *external transition*. In the case an external event arrives at the exact time of an internal event, then the confluent state transition function $\delta_{con}(s, x)$ determines the next state; either an internal transition followed by an external transition or an external transition followed by an internal transition, or something completely different.

A coupled DEVS formalism corresponds to structure systems in the systems knowledge hierarchy where a set of systems and their interrelations are specified (§ 2.1.3 and § 2.2.1). DEVS models are closed under coupling; that is, the hierarchical DEVS models can always be expressed as an equivalent atomic model (Vangheluwe, 2000). In other words, a hierarchical DEVS model theoretically behaves like, and has exactly the same characteristics as an atomic model (which does not mean that we know or can always define that atomic model). Therefore, in hierarchical model composition, a component in a hierarchical model can also be another hierarchical model, since this model has (and is indistinguishable from) an equivalent atomic model. This allows for an infinite hierarchy of

hierarchical models, where on each level, atomic sub-models and hierarchical sub-models can co-exist.

The formal definition for a coupled DEVS model as a structure N is described in Definition 2.3.

Definition 2.3. Formal definition of a coupled DEVS model (based on Zeigler et al., 2000)

$$N = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}\ select >$$

where

X is the *set of input events* of the coupled model

Y is the *set of output events* of the coupled model

D is the *set of component indexes* of the coupled model

$\{M_i\}$ is a set of components defined as

$$M_i = < X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i >, \forall i \in D$$

$\{I_i\}$ for each $i \in D$, I_i is the *set of components* which are influenced by component i, and $I_i \subseteq D \cup \{N\}$, $i \notin I_i$

$\{Z_{i,j}\}$ for each $j \in I_i$, $Z_{i,j}$ is the *output-to-input translation function*, where

$$\begin{cases} X \rightarrow X_i & \text{if } i = N \text{ and } j \in D \\ Y_i \rightarrow Y & \text{if } i \in D \text{ and } j = N \\ Y_i \rightarrow X_J & \text{if } i \in D \text{ and } j \in D \end{cases}$$

select $2^D \rightarrow D$ is the tie-breaking function which defines how to select the event from the set of simultaneous events. In other words, when multiple (atomic) models have to change their state at the same time, Select determines the order in which the (atomic) models are allowed to change their state one by one.

In the port-based definition of DEVS (Wainer, 2009; Zeigler et al., 2000), model components are modeled to have a set of input and output ports. The message-based communication structure of the port-based DEVS formalism guarantees the modularity among the model components through their input and output ports. Ports also enable

setting up the coupling relationship between model components. The classic DEVS atomic model with the port specification is specified in Definition 2.4:

Definition 2.4. Specification of a port-based DEVS atomic model (based on Zeigler et al., 2000)

$$M = < X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >$$

where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$	is the set of *input values* and *ports*
$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$	is the set of *output values* and *ports*
S	is the *set of sequential states*
$\delta_{ext}: Q \times X^b \longrightarrow S$	is the *external state transition function*, where $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the *set of total states*, e is the elapsed time since last state transition
$\delta_{int}: S \longrightarrow S$	is the *internal state transition function*
$\delta_{con}: S \times X^b \longrightarrow S$	is the *confluent state transition function*
$\lambda: S \longrightarrow Y^b$	is the *output function*
$ta: S \longrightarrow \mathbb{T}^\infty$	is the *time advance function* which is used to determine the lifespan of a state, where $\mathbb{T}$ is the time base and $\mathbb{T}^\infty = [0, \infty]$ is the set of non-negative real numbers plus infinity

In the above port-based atomic DEVS model description, the superscript b indicates that the input and output values can be a bag[6] of values rather than a single one. Therefore, X^b indicates a set of bags over elements in X where $\delta_{ext}(s,e,x) = s$. The set of total states

[6] A bag is a set with possible multiple occurrences of its elements.

(S) that a model has is the Cartesian product of sets (s) belonging to each state variable. This means $S = s_1 \times s_2 \times \ldots \times s_n$, where $n \in \mathbb{N}$.

Similarly, a DEVS coupled model can be described using a port specification, as provided in Definition 2.5.

Definition 2.5. Specification of a port-based DEVS hierarchical model (based on Zeigler et al., 2000)

$$M = < X, Y, D, \{M_{d \in D}\}, EIC, EOC, IC, select >$$

where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$	is the set of (external) *input ports* and *values*
$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$	is the set of (external) *output ports* and *values*
D	is the *set of component names*
$\{M_{d \in D}\}$	is the *set of components*, each of which can be either an atomic model or a coupled model
$EIC \subseteq X \times \cup X_{d \in D}$	is a set of *external input couplings* that connects external inputs to components inputs, where $X_{d \in D}$ is the set of inputs of $M_{d \in D}$
$EOC \subseteq \cup Y_{d \in D} \times Y$	is a set of *external output couplings* that connects component outputs with external outputs, where $Y_{d \in D}$ is the set of outputs of $M_{d \in D}$
$IC \subseteq \cup Y_{d \in D} \times \cup Y_{d' \in D \wedge d \neq d'}$	is a set of *internal couplings* that connect component outputs to component inputs
select	$2^D \rightarrow D$ is the *tie-breaking function* which defines how to select the event from the set of simultaneous events.

The hierarchical specification for DEVS is important from a complexity point of view (hierarchical modeling) and therefore specifically important for large-scale complex systems models.

2.3 Temporal Data Mining: An Overview

As argued in §1.3.4, one of the challenges related to large-scale complex simulation models are the growth in the volumes of data generated by their executions, particularly state-trace data, or state-sequence data. *Data mining* is the process of discovering knowledge, such as patterns, associations, trends, anomalies, and significant structures, from such large and complex data sets using a combination of techniques from different fields such as statistics, machine learning, pattern recognition, database, and high-performance computing technologies (Fayyad et al., 1996a; Hand et al., 2001; Laxman & Sastry, 2006). Typical data mining tasks include *preprocessing*, *clustering*, *segmentation*, *classification*, *regression*, and *association rule mining* (Dunham, 2002; Han et al., 2012; Hand et al., 2001).

(I) *Preprocessing* the data mainly includes the tasks of data cleaning (i.e., removal of outliers, erroneous, missing, or irrelevant data), data integration (i.e., combining data from multiple sources into one), data transformation (i.e., conversion of data values to a format required by the tools and algorithms; for example, quantization of the continuous data points or conversion of numerical values to categorical ones (Mörchen, 2006b), and normalization (i.e., adjusting different data values to a common scale without distorting the differences in ranges of values).

(II) *Clustering*[7] is the process of identifying intrinsic regions or classes (referred to as clusters) embedded in a given data set based on similarity measure, such as distance (i.e., the distance of small intra-clusters vs. the distance of large inter-clusters) or density (i.e., dense cluster regions separated by comparably sparser regions; Mörchen, 2006b). K-means (MacQueen, 1967) and DBSCAN (Ester

[7] Also referred to as *unsupervised classification* (Rui & Wunsch, 2005).

et al., 1996) are two of the widely used clustering algorithms (see Benabdellah et al., 2019; Berkhin, 2006; Rui & Wunsch, 2005 for a collection of data clustering methods).

(III) *Segmentation* is the task of partitioning a large data set into smaller portions or segments (Shani et al., 2011). The major difference between segmentation and clustering is that segmentation uses borders, or cut points, to split the whole data set into smaller segments, whereas clustering aims to identify borders within a given data without the goal to assign all points to clusters (Gionis & Mannila, 2003; Lovric et al., 2014). For example, clustering algorithms can label data points as noise in a given data set and do not assign them to any of the segments (Berkhin, 2006). Segmentation techniques are commonly used in discovering meaningful boundaries within long sequences of univariate data (see Keogh et al. (2004) for a comprehensive review of such segmentation methods) and multivariate data (Mäntyjärvi et al., 2001; Mörchen, 2006b; Siskind, 1999).

(IV) *Classification* is the task of assigning data items within a given input data set into predefined classes or target categories according to a classification model learned from training data (Hand et al., 2001). In classification, the input data set is divided into two parts: a training set, which is used in the generation of the classification model, and a validation set, which is used to measure the accuracy of the *classifier*. A classifier is an algorithm, or a mathematical function, which is implemented for the classification task (see Witten & Frank, 2002 for a collection of classification methods). Classification can be seen as a particular version of regression, where the predictor variables are discrete and with no implicit ordering, instead of being numerical (Mörchen, 2006b). A recent literature review of text classification algorithms can be found in Li et al. (2022).

(V) *Regression* is the task of predicting a range of sub-variables (i.e., response variables) from the numerical values of known variables (i.e., predictor variables) using explicit variable dependencies (Hastie et al., 2009). Regression methods are also useful to replace missing values in a data set. Linear (Yan &

Su, 2009) and non-linear methods (Goodarzi et al., 2009; Seber & Wild, 2003) are two of the widely known forms of regression.

(VI) *Association rule mining* aims to discover correlations, frequent patterns, associations or causal structures among data items within various types of databases such as transactional, relational, and other types of data repositories (Huang et al., 2000). Typically, association rules are calculated for itemsets (i.e., discrete set of items) and are used to express the co-occurrence of data items in these itemsets. An itemset is considered as *frequent* if all subsets of an itemset are frequent (Mörchen, 2006b). A particular subfield of association rule mining that deals with the discovery of frequent patterns within large itemset databases (e.g., customer transaction database) is called frequent itemset mining (Agrawal & Srikant, 1994). Apriori (Agrawal & Srikant, 1994) is the best-known association rule mining algorithm (see Zhao & Bhowmick (2003) for a survey of association rule mining algorithms).

Extensive overviews on the methods and applications of data mining are available (Fayyad et al., 1996a, 1996b; Han et al., 2012; Hand et al., 2001; Kantardzic, 2011; Mannila, 1997).

2.3.1 Temporal Data Mining: Definition and Data Types

Temporal data mining is an extension of data mining with a particular focus on the discovery of knowledge from data with temporal aspects (Fu, 2011; Grossmann & Rinderle-Ma, 2015; Laxman & Sastry, 2006; Mitsa, 2010). Temporal data is typically represented as sequences of observations (generally numerical or categorical values) at discrete points in time (Mamoulis, 2009). In cases where time is not uniformly sampled, there is usually some lower bound for the granularity (i.e., finest level of detail) of time which is referred to as *time points* (Mörchen, 2007).

Temporal data mining methods are designed to analyze mainly four types of temporal data, each of which consists of numerical or categorical values, and can be represented as univariate or multivariate (Fu, 2011; Grossmann & Rinderle-Ma, 2015; Laxman & Sastry, 2006; Lin et al., 2002; Mitsa, 2010; Mörchen, 2007). A *time series* is a series of data points indexed by equidistant points in time (Laxman & Sastry, 2006; Mörchen, 2007).

Time series are *univariate* if they contain only one time-dependent variable (i.e., consisting of a single series), or *multivariate* if they have more than one time-dependent variable (i.e., consisting of multiple series). A time series is numerical if the values at each time point is represented with numerical values (e.g., stock ticks, EEG, temperature), or *categorical* if the values of each time point is a category (e.g., timed-event logs).

The second type of temporal data is a *sequence,* also referred to as event sequences (Mamoulis, 2009) or transactions. A *sequence* is a series of timed events ordered by qualitative temporal concepts, such as *position* like *before* or *after* (Allen, 1983; Das et al., 1998; Mannila et al., 1997; Mörchen, 2006b), instead of an explicit equidistant time indexing as it is the case in time-series. In other words, if an item appears before another one in such transactional data, it only indicates that the former has occurred before the latter (without guaranteeing any equidistant time separation between them or between any other successive events). Examples of sequence data could be a text, gene or protein sequences, or a list of moves during a chess game. In some cases, some of these items can be deemed meaningful when they repeatedly occur together in a given sequence data. When identified, these set of items can be used to describe the behavior and actions of users or systems, and predict future items. An *event sequence* is a special type of sequence data, consists of events ordered by non-equidistant points in time. If an event sequence only consists of non-simultaneous events, it is referred to as *simple event sequence*. On the other hand, an event sequence with simultaneous events is referred to as *complex event sequence*. An *episode* can be described as a sequence of events, where each event has an associated time of occurrence (Mannila et al., 1997). A frequent episode, in this context, can be described as an episode appearing at least at a user-defined frequency threshold in an event sequence data set (Mannila et al., 1997).

The third type of temporal data is an *interval time series*, which is a set of time intervals recorded at each time point instead of a single value; e.g., a yearly record of the highest and lowest temperature in Amsterdam for each day, or the daily range of sea level at various locations (Mörchen, 2006a; Villafane et al., 1999).

Lastly, an *itemset sequence* is a time-ordered sequence of a non-empty set of (unordered) items (Agrawal & Srikant, 1995). An example of an itemset sequence can be a customer transaction database, where each transaction is an ordered collection of purchased items

by a customer at each visit. The process of exploring the relationships between purchased products that frequently appear together, also known as *market basket analysis*, is a typical example of frequent pattern mining (Han et al., 2012).

2.3.2 Characteristics of State-Trace Data

In our research, the input data for the temporal data mining tasks are the state-trace outputs generated from the execution of large-scale complex DEVS models. State-trace data obtained from such models can have the characteristics of a sequence or multivariate time series consisting of categorical variables, numerical variables, or both.

Definition 2.6. *A state-trace of a discrete-event simulation model is a time sequence of state-trace records (recorded instants), where each state-trace record within the same state-trace data set is a homogenous (i.e., each state-trace record in the state-trace data has the same number variable values) and has a total order (i.e., the model variable values from with the same index create a column and all values in a single column represents the same model variable).*

A state-trace data set can have variables with all categorical, all numerical, or hybrid variables columns. Depending on the goals and the objectives of the simulation study and the preferences of the modeler, input variables, output variables (e.g., run statistics) of the simulation model and the time (e.g., elapsed, absolute, simulation) can be included as an individual column in the state-trace data. The changes between two consecutive state-trace records describe the system state changes at a rate (i.e., at every event occurrence or at a fixed-increment of time) imposed by the modeler. As a result, the size of a state-trace data set is a combination of (a) the number of rows determined by the run length and logging rate, and (b) the number of columns determined by the number of state variables, input and output variables, time, and any additional data required by the modeler.

DEVS simulators are event-driven at simulation time. Generally, the simulation time jumps to the occurrence of the next event using the *ta* function or until the next external event, assuming that there is no change in the system between two consecutive events (Zeigler et al., 2000). This approach is called *next-event time progression*. As a result, state-trace records are generated only at the time an event occurs, and time is, therefore, non-uniformly sampled. In that case, either the time of event occurrences or the elapsed time

between events has to be part of the state-trace data in order to preserve the temporality of the data. Alternatively, a *fixed-increment time progression* approach can be employed, in which time is sliced into uniform (equal) timeframes and the state of the system is represented by the convolution of events happening in each timeframe. With the fixed-time progression approach, successive state-trace records will be the same if no event occurs in between, and state changes will not be recorded when multiple events take place between updates of the state trace records. In this research, we will show the results for both approaches.

As mentioned earlier in §2.2.4.1, DEVS model components (both atomic and coupled) have state variables that, as a collection S, indicate the state of the system at any point in time.

2.3.3 Temporal Data Mining Tasks

The mining tasks that we described earlier in §2.3 can be applied on temporal data, most of which are direct extensions of the corresponding mining tasks on general types of data (Mamoulis, 2009). In addition, there are several mining algorithms that are specific to temporal data (Fournier-Viger et al., 2017; Gan et al., 2019; Han et al., 2012; Laxman & Sastry, 2006; Mitsa, 2010; Roddick et al., 2001). The two major ones are *sequential pattern mining* (SPM; Mabroukeh & Ezeife, 2010; Pei et al., 2001) from sequential databases, and *frequent episode mining* (FEM; Mannila et al., 1997) from long temporal data (e.g., event) sequences.

SPM deals with the discovery of sequential patterns, or subsequences, ordered by a temporal concept such as time or position within a given sequential dataset (Laxman & Sastry, 2006). Such patterns can be differentiated from the other possible patterns in the data based on various criteria such as their occurrence frequency and length (Fournier-Viger et al., 2017; Gan et al., 2019). Two types of data that are often used in SPM are time-series (numerical and categorical) and sequences. Numerous algorithms have been developed to identify sequential patterns in sequence datasets (Fournier-Viger et al., 2017; Gan et al., 2019; Mabroukeh & Ezeife, 2010). Although these various algorithms generally produce the same output if the input parameters are the same, they utilize different strategies and data structures to discover sequential patterns (Fournier-Viger et al.,

2017; Gan et al., 2019). SPM algorithms can be distinguished on the basis of four characteristics: (a) whether they use *depth-first search* (i.e., patterns are considered by exploring a node branch as far as possible before backtracking and expanding other nodes) or *breadth-first* search (i.e., patterns are considered in ascending order of length), (b) whether they use a horizontal (level-wise) or vertical (i.e., on the basis of itemsets) database representation, (c) how they determine the next patterns to be explored, and (d) how they determine the minimum support (i.e., a user-defined frequency threshold) criteria (Fournier-Viger et al., 2017).

The goal of FEM is to discover frequent episodes from single long temporal data (or event) sequences or to discover episode rules between events describing which event (or a sequence of events) often appears before another event within a user-defined time frame (Huang & Chang, 2008). The particular interest in discovering patterns from a single sequence rather than a set of sequences is the main difference between SPM and FEM (Fournier-Viger et al., 2017; Gan et al., 2019). Based on the categorization of Mannila et al. (1997), episodes can be *parallel* (i.e., the set of events occurring within a window of time, but not in a particular order) or *serial* (i.e., a list of events occurring in total order within a window of time) or *composite* (i.e. a combination of serial and parallel episodes). An episode is considered frequent if it appears above a user-defined frequency threshold called *minimum support threshold* or *minsup*. However, choosing the minsup value is a difficult and time-consuming task, because of the fact that setting a high or a low minsup value is a trade-off between long execution times and insufficient candidate pattern set generation to successfully unearth important information, and determining an "enough but not too many" candidate set is done via trial and error in traditional FEM algorithms (Fournier-Viger et al., 2020).

The WINEPI and MINEPI algorithms were the first algorithms for FEM (Mannila et al., 1997). The WINEPI algorithm (Mannila et al., 1997) mines all frequent episodes (parallel or serial) using a breadth-first search and placing a sliding window over the input sequence. The algorithm counts the frequencies of nodes of increasing length (up to the user-defined window size) until the window reaches the end of the input sequence. Another *window-based frequency* algorithm is MINEPI (Mannila et al., 1997), which also utilizes a breadth-first search approach but only counts the minimal occurrences of episodes.

However, the apriori-like candidate generation characteristic of window-based frequency algorithms has the limitation of missing some of the frequent episodes with longer length as they do not hold an anti-monotonic property and the requirement to keep the candidate patterns in memory because of the breadth-first search is costly (Ao et al., 2015). Another shortcoming of window-based frequency algorithms is that some occurrences of episodes may be counted more than once due to their support count mechanism (Fournier-Viger et al., 2020). A commonly used user-defined input parameter for window-based frequency algorithms is called *maximum window bound* or *maxwin*. Given a large data set, mining frequent episodes above a given minsup might yield results that span across too many intervals. To mitigate that, users can thus use the maxwin parameter to mine only the episodes that are less than or equal to the maxwin interval (Huang & Chang, 2008).

To address these limitations of window-based frequency algorithms, two new depth-first search based algorithms MINEPI+ and EMMA (Episodes Mining using Memory Anchor) were proposed (Huang & Chang, 2008). Specifically, the EMMA algorithm was shown to outperform MINEPI and MINEPI+ (Huang & Chang, 2008) as the algorithm utilizes both depth-first search and memory anchors to further reduce the search space in pattern generation and to accelerate the mining task. In recent years, several other studies have been published in the field of FEM to tackle problems, such as finding top-k episodes in an event sequence (Fournier-Viger et al., 2020), mining episodes on dynamic event streams (Patnaik et al., 2012), online frequent episode mining (Ao et al., 2015), and high-utility episode mining (Fournier-Viger et al., 2017; Gan et al., 2019; Lin et al., 2017; Truong-Chi & Fournier-Viger, 2019; Wu et al., 2013), which focuses on providing other domain-specific metrics to select frequent episodes instead of the *minsup* parameter used in traditional FEM algorithms.

2.3.4 Temporal Data Mining in Discrete Event Simulations

Following the above discussion of data mining concepts and techniques, there are several reasons why data mining, and specifically temporal data mining, may form an essential contribution to the study of large-scale complex simulation models.

Firstly, as discussed in the previous chapter, the manual abstraction of large-scale complex simulation models at the structure level is a near-impossible endeavor due to the vast and diverse number of objects and the relations between them, and currently, no strategy exists yet to automate this process. An alternative approach to model the abstraction of large-scale complex simulation models is to simplify the dynamic behavior that is encapsulated in the state-trace data of these models. However, state-trace data obtained from these simulation models are extensive in terms of volume and variety, which confines modelers' ability to identify and utilize frequent patterns for model abstraction. Temporal data mining tasks, such as sequential pattern mining and frequent episode mining, can offer a set of tools to automate this model abstraction process at the transformation level, theoretically corresponding to the generative system level in Klir's (1985) system knowledge framework and the state transition level in Zeigler's System Specification (Zeigler et al., 2000).

Secondly, as discussed in §2.2.4, it is often impossible to fully capture the totality of large-scale complex systems in a single simulation model (Hofmann, 2004; Yilmaz & Ören, 2004). A family of models with different levels of abstraction, or resolution, can yield different types of insight into the dynamic behavior of large-scale complex systems, to provide better support for users with different roles or perspectives. Unfortunately, the previously mentioned shortcomings for the manual abstraction of large-scale complex simulation models remains in the existing multiresolution modeling techniques (Yilmaz & Tolk, 2006). Most temporal mining algorithms have various user-defined parameters, including window size and the maximum time duration, which can be used to predetermine the minimum size of the frequent patterns to be detected by the algorithms. Such parameters can serve as a "resolution slider" to increase (not beyond the resolution of the state variables in the base model) or decrease (not beyond the minimum threshold to sustain the model validity) the resolution of the model. In addition, modelers can apply temporal data mining tasks only on a select set of state-traces (e.g., those that belong to certain atomic or coupled models that are contextually linked) from the complete state-trace data. This would help modelers to decrease the resolution of a specific portion of the model (similar to the concept of "zooming out") and preserve the details for the remainder (of the model).

To our knowledge, no studies to date have applied temporal data mining techniques on state-trace data collected from large-scale complex discrete-event models to build homomorphic lumped models. However, the general idea of inferring model structure from data has previously been explored in the field of process mining. Process mining has been applied with promising results in a wide range of domains, including healthcare, information and communication technology, manufacturing, education, and logistics (Garcia et al., 2019). Process mining provides techniques to discover, monitor, and enhance business processes based on event logs. The purpose of process discovery in process mining is to extract knowledge from event logs with the goal of generating a process model, which is usually a Petri net model (van der Aalst, 2011, 2016). A Petri net (Dennis, 2011; Narahari, 1999; Petri & Reisig, 2008) is a directed bipartite graph with transitions and places connected with directed arcs. For instance, Lugaresi & Matta (2021) propose a process mining-based method for generating and tuning both Petri net and simulation models from the event logs of manufacturing systems. In (Lugaresi & Matta, 2022), they also investigated the applicability of Digital Twins approach for online model generation of discrete-event simulation models from event logs. Digital Twins is a rapidly growing field (Liu et al., 2021). We believe that Digital Twins as an application area could benefit from the concept of state-trace mining, and help to create and test new techniques for state-trace mining.

2.4 Summary and Outlook

In this chapter, the fundamental concepts and the existing work in systems theory, modeling and simulation (with a particular focus on discrete event simulation), and temporal data mining were introduced. Furthermore, we described the characteristics of the state-trace data generated from the execution of large-scale complex DEVS models. Finally, we discussed why temporal data mining tasks on state-trace data can be a useful tool for the automation of DEVS model abstraction and the generation of multiresolution models. However, as we will discuss, the application of temporal data mining procedures in discrete event simulations introduces several unique theoretical and practical issues. In the next chapter, we propose a method based on temporal data mining for DEVS model abstraction that addresses these specific issues and considerations.

CHAPTER 3

Temporal Data Mining-based Method for Automated Discrete-event Model Abstraction

3 Temporal Data Mining-based Method for Automated Discrete-event Model Abstraction

As we have established in the previous chapters, conventional methods for model abstraction at the structure level (Zeigler et al., 2000) are not suitable for large-scale complex simulation models. Instead, abstraction at the transformation level, using the state-trace data which encapsulate the model's dynamic behavior, may be a more viable alternative approach for abstracting such models. Given its ability to automate the detection of frequent temporal patterns in large data, temporal data mining may provide useful and versatile tools for this process. In this chapter, we integrate these ideas and present a novel temporal data mining-based method for DEVS model abstraction using state-trace data. DEVS (Zeigler et al., 2000) is the universal common modeling formalism for the simulation of large-scale complex simulation models (Vangheluwe & de Lara, 2002). In what follows, we will provide a detailed description of our proposed method, specifically for the field of simulation. We will highlight the strengths of this novel approach and introduce several theoretical and practical considerations in its application. We will formally describe the key concepts and algorithms used in our method, and present step-by-step guidelines for its application.

3.1 Introduction

Our proposed method integrates the fields of temporal data mining and modeling and simulation. In essence, the goal of our method is to automate the process of model abstraction in dynamic DEVS. More specifically, it aims to generate a simplified (lumped) homomorphic DEVS model from a larger-scale, more complex simulation model (base model). This is achieved by utilizing the state-trace data of the base model and by employing temporal data mining tasks to automatically detect frequent temporal patterns. These frequent temporal patterns, in turn, will become the aggregated states of the lumped model. Subsequently, the lumped model will be formally described using Markov Modeling (i.e., Markov Chain), which is one of the commonly used techniques to describe probabilistic/stochastic models (Mor et al., 2020). In addition, its concepts of states and state transitions are fully compatible with DEVS description of discrete event

systems, making it a useful tool to address challenges resulting from large-scale complex simulation models (Seo et al., 2018; Zeigler et al., 2018). The Markov Chain will then be simulated to generate output in the form of new state-trace data. Finally, the state-trace data of the lumped Markov Chain will be validated against the state-trace data of the base model. The rationale for the above choices will be elaborated in the course of this chapter.

3.1.1 Formalization of Discrete-event State-Traces

Recall from Chapter 2 that the DEVS formalism allows describing systems as a composition of *atomic* (behavioral) and *coupled* (structural) models. Atomic models are expressed in the basic DEVS formalism and atomic model components can be coupled to build more complex coupled models. These coupled models are expressed in the coupled DEVS formalism and can be used as components of larger coupled models, allowing a hierarchical model description (Zeigler et al., 2000).

The basic DEVS formalism contains information about the state transition of a system as it corresponds to systems knowledge at the generative level and systems specification at the state transition level (§2.1.3 and §2.2.1). Therefore, the behavior of an atomic DEVS model can be described as a set of sequences of timed events, i.e., a sequence of deterministic transitions between sequential states over time at the generative level (Vangheluwe, 2000).

The state of a component (atomic or coupled) remains constant over intervals of time. A state s_i is *passive* when its lifetime is infinite (i.e., $ta(s_i) = \infty$), and *active* when its lifetime is a finite positive number. If S_a is a subset of active states and S_p a subset of passive states, $S_a \cap S_p = \emptyset$. The values of the component states only change at predetermined *event times*. An *event* represents a change in the state. A transition function in DEVS is decomposed into two sub-functions – the internal transition function that specifies the state changes caused by internal events and the external transition function that specifies the state changes due to external events (see §2.2.4.1 for more details). The total states (s, e) of a model can be defined as:

$$TS = \{(s, e) | s \in S, 0 \leq ta_i(s_i)\}$$

where e is the elapsed time in state s (Zeigler et al., 2000). This concept of total states is crucial as it enables one to specify a future state on the basis of the elapsed time in the current state (Giambiasi & Frydman, 2014).

Because DEVS models are closed under coupling, any DEVS model, whether atomic or coupled, can be flattened with an equivalent atomic DEVS model (Vangheluwe, 2000). This means that even more complex models like large-scale complex simulation models (i.e., structural models) can be fully specified by a simpler atomic (i.e., behavioral) DEVS model.

Consider a coupled DEVS component $N = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select >$, where $\forall i \in D$, M_i is an atomic component defined as $M_i = < X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i >$ (see §2.2.4.1 for more details). For any atomic component M_i, a state change is triggered by the DEVS simulator executing internal state transitions $\delta_{int,i}(s_i)$ and external state transitions $\delta_{ext,i}(s_i, e_i, x_i)$. State transitions of the coupled DEVS model N are driven by state transitions of its atomic components M_i. Therefore, the coupled DEVS model N is equivalent to an atomic DEVS model $M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$ (Vangheluwe, 2000). The resultant set of sequential states of M (equivalent to the coupled DEVS model N) is the product of all the total state sets of all the components

$$S = (\ldots, (s_i, e_i), \ldots) \in S = \times_{i \in D} Q_i$$

$$\text{where } Q_i = \{(s_i, e_i) \mid s_i \in S_i, 0 \leq e_i \leq ta_i(s_i)\} \text{ and } ta: S \rightarrow \mathbb{R}^{+}_{0,+\infty}$$

For the coupled model, an internal state transition to the sequential state $\delta_{int,i^*}(s_{i^*})$ is triggered from an internal state transition of the selected imminent component i^* (Vangheluwe, 2000), which transforms the different parts of the total state as follows:

$$\delta_{int}(s) = (\ldots, (s'_i, e'_i), \ldots) \quad \text{, where}$$

$$(s'_i, e'_i) = (\delta_{int,i}(s_i), 0) \quad \text{, for } i = i^*$$

$$= (\delta_{ext,i}(s_i, e_i + ta(s), Z_{i^*,i}(\lambda_{i^*}(s_{i^*}))), 0) \quad \text{, for } i \in I_{i^*}$$

$$= (s_i, e_i + ta(s)) \quad \text{, otherwise}$$

$$\text{where } ta(s) = min\{ta_i(s_i) - e_i \mid i \in D\}$$

The external transition function transforms the different parts of the total state as follows:

$$\begin{aligned} \delta_{ext}(s, e, x) &= (\ldots, (s'_i, e'_i), \ldots), \text{ where} \\ (s'_i, e'_i) &= (\delta_{ext,i}(s_i, e_i + e, Z_{N,i}(x_i)), 0) && \text{, for } i \in I_N \\ &= (s_i, e_i + e) && \text{, otherwise.} \end{aligned}$$

The key element in the closure procedure is the selection of the most imminent event from all scheduled events belonging to all components forming the coupled model. In case a number of imminent events are scheduled simultaneously, the *select* function is used as a tie-breaker. The closure under coupling property is important for our research because it enables us to define the state-trace of a coupled model where all events are ordered sequentially with respect to the time base of the model.

As defined earlier in Definition 2.6, a state-trace of a discrete-event simulation model is a time sequence of state-trace records (recorded instants), where each state-trace record within the same state-trace data is a homogenous (i.e., each state-trace record in the state-trace data has the same number of variable values) and has a total ordering (i.e., the model variable values from with the same index create a column and all values in a single column represents the same model variable). Depending on the goals and the objectives of the simulation study and the preferences of the modeler, input variables, output variables (e.g., run statistics) of the simulation model and the time (e.g., elapsed, absolute, simulation) can be included in a state-trace record.

3.1.2 Frequent Episode Mining

In §2.3.3 and §2.3.4, we described how temporal data mining may provide an essential contribution to the study of large-scale complex simulation models and we discussed the importance of finding frequent behavioral patterns in large temporal data (e.g., state-trace data) to facilitate the process of model abstraction. The state transition mechanism and the time advance structure of DEVS characterization of a discrete event systems are fully compatible with the concepts of states and state transitions in Markov Modeling (Zeigler et al., 2019). In DEVS, the next state can be determined solely by knowing the current state and the elapsed time since the last state transition. Therefore, the dynamic

behavior of a DEVS model encapsulated in state-trace data can be simplified by (a) identifying all successive state-trace record pairs (i.e., serial episodes of length two) corresponding to the system of interest's state-transitions by the frequent episode mining algorithm and (b) calculating the state transition probabilities for these identified pairs. In our method, we employ the EMMA (Huang & Chang, 2008) frequent episode mining algorithm to discover all serial[8] state-trace record pairs from the state-trace data and their frequency information (also see §2.3.3 for more details on the EMMA algorithm). In all our case studies, we used the Java implementation of the EMMA algorithm provided within the SPMF open-source data mining library (Fournier-Viger et al., 2014; 2016). We chose the SPMF library because it provides source code that is widely used in the data mining field, well-documented, and modular (i.e., that can be easily integrated and extended). In addition, the version of the EMMA algorithm implemented in the SPMF library[9] contains all the optimization methods described in (Huang & Chang, 2008).

Although several frequent episode mining algorithms (see §2.3.3) can be used for our purpose to find serial episodes, we chose EMMA as it is one of the most efficient FEM algorithms (Fournier-Viger et al., 2020; Huang & Chang, 2008). Specifically, Huang and Chang (2008) have shown that EMMA outperforms other well-known traditional FEM algorithms, such as MINEPI and MINEPI+. Moreover, as we show in Appendix A.1, a comparison of these three algorithms on our case study data confirms that, at the time of this research, EMMA is the fastest in finding all unique state-trace records and consecutive state-trace record pairs from state-trace data of various length (5,000; 10,000; 25,000 and 50,000 state-trace records).

In Fournier-Viger et al. (2020), researchers have proposed a new algorithm named TKE to find top-k frequent episodes. Because this algorithm is so new, it was outside the scope of the current research to test it on our data set. However, the study's findings on

[8] We acknowledge that the EMMA algorithm is also capable of discovering partially-ordered parallel episodes. However, the focus of our research is to identify serial episodes from state-trace data as described in §3.1.1. Therefore, we will limit the algorithm's discovery set by adjusting its *minsup* and *maxwin* parameters and, for the rest of the dissertation, the term *episode* will be used to refer serial episodes.

[9] Specifically, we used SPMF release version 2.41, which was the latest release of the library at the time being.

benchmark datasets suggest that the algorithm may be a valuable alternative to EMMA when the user lacks sufficient background information about a dataset to correctly set the optimal *minsup* threshold. Future studies should compare the performance of EMMA with the new TKE algorithm on state-trace data.

The EMMA algorithm consists of 3-stages. At stage 1, the algorithm aims to mine all frequent serial episodes to construct frequent serial episodes. This is to (a) avoid checking items more than one time, (b) encoding each frequent itemsets with a unique ID, such as time or order to construct them into an encoded horizontal database (c) mining the frequent serial episodes in the encoded horizontal database; we refer to Huang and Chang (2008) for more details. Algorithm 3.1.2.1 describes in detail stage 1 of EMMA; the application of the *frequent itemset mining using memory anchor* (FIMA) algorithm.

Algorithm 3.1.2.1. EMMA stage 1: FIMA (Huang & Chang, 2008)

Procedure of **FIMA** (**temporal data** *TDB*, *minsup*)

1: **Scan** *TDB***, find frequent 1-item** F_1;

2: **Remove nonfrequent items and transform** *TDB* **into indexed database** *IndexDB***;** *meanwhile* **maintain the locations of all** F_1 **in the flat dataset**;

3: **for each** f_i in F_1 **do**

4: **Output** f_i **and its TidList;**

5: **fimajoin**(f_i, f_i.LocationList)**;**

Subprocedure of **fimajoin(***Pattern, LocationList***)**

6: ***LFI* = local frequent items in the projected location list of** *FP* (i.e., FP. PList)**;**

7: **for each** lf_j **in** *LFI* **do**

8: Output *FP* ∪ lf_j **and its TidList;**

9: **fimajoin(***Pattern* ∪ lf_j , lf_j.*LocationList*)**;**

At stage 1, the goal of the EMMA algorithm is to utilize the frequent itemsets mining algorithm FIMA to eliminate the unnecessary generation of candidate items. The FIMA algorithm achieves this by validating local frequent items and subsequently, reducing the computation. The recursive call of the sub-procedure *fimajoin* stops when no further frequent itemsets can be generated. Stage 2 and stage 3 of the EMMA algorithm are described in Algorithm 3.1.2.2.

Algorithm 3.1.2.2. EMMA stage 2 and 3 (Huang & Chang, 2008)

Procedure of **emmaassociate** (**temporal data** *TDB*, *minsup*, *maxwin*)

1: **Call FIMA**(*TDB, minsup*) **to find all frequent items** FP_1 **and their Tid-Lists;**

2: **Associate each item with a unique** ID **to construct an encoded database** *EDB*;

3: **for each** fid_i **in frequent** *ID***s** FP_1 **do**

4: **Output** fid_i**;**

5: if (*ExtCount*(fid_i.*boundlist*) ≥ *minsup* * |*TBD*|)

6: **emmajoin**(fid_i, fid_i.boundlist)**;**

Procedure of **emmajoin(***Episode, boundlist***)**

7: *LFP* **= local frequent IDs in the projected bound list of** *Episode* (**i.e.,** *Episode.PBL*)**;**

8: **for each** lf_i **in** *LFP* **do**

9: **Output** *Episode* · lf_i**;**

10: *tempBoundlist* = *temporalJoin(boundlist,* f_i.*boundlist)***;**

11: **if** (*ExtCount(tempBoundlist)* ≥ *minsup* * |*TBD*|)

12: **emmajoin**(*Episode* · lf_i,*tempBoundlist*)**;**

The second stage of the EMMA algorithm aims to associate each frequent item with a unique identifier. These associations are stored in the encoded database *EDB*. Once the frequent itemsets are generated for the given *minsup*, the EMMA algorithm starts

associating each frequent itemset with a unique ID and constructs an encoded horizontal database (Huang & Chang, 2008). At stage 3, the main task is to mine frequent episodes that are limited by *maxwin* parameter to define the projected bound list (PBL) for an episode. The PBL allows detecting the frequent IDs by counting the number of bounds that an ID occurs (Huang & Chang, 2008, p.106).

3.1.3 Markov Chains

A stochastic process $X = \{X_n : n \geq 0\}$ on a discrete (i.e., finite or countable) set of all possible sequential states, or *state space*, S is a Markov Chain (Serfozo, 2009) if, $\forall\ i, j \in S$ and $n \geq 0$,

$$\mathrm{P}\{X_{n+1} = j \,|X_n = i,\ X_{n-1} = i_{n-1}, \dots, X_0 = i_0\} = P\{X_{n+1} = j|X_n = i\} \tag{3.1}$$

$$\mathrm{P}\{X_{n+1} = j|\ X_n = i\} = P_{ij} \tag{3.2}$$

A Markov Chain is a discrete-time Markov Chain if the state transitions happen at discrete times $n \in \mathbb{N} = \{0, 1, 2, \dots\}$ (Sorensen & Gianola, 2007). X_0 is denoting the initial state. The value of X_n is the state of the process at time n (e.g., if $X_n=i$, the process is said to be in state i at time n). π_0 is the initial distribution of the Markov Chain at t_0. π_0 can be interpreted as a row vector, also referred to as *initial distribution vector*, whose ith entry $\pi_0(i)$ can be denoted as the probability $\mathbb{P}\{X_0 = i\}$ that the chain starts in state i.

Condition (3.1), called the *Markov property*, says that, at any time n, the next state X_{n+1} is conditionally independent of the past states $X_0, \dots, X_{n-1}$ and just dependent on the present state X_n. This *memoryless* property of Markov Chain states is akin to the discrete-event model states in DEVS and forms the backbone of our temporal data mining-based method to abstract discrete-event simulation models.

Condition (3.1) also indicates that the Markov Chain is *time-homogenous*, that is the transition probabilities are independent of the time parameter n, but rather constant over time (Serfozo, 2009). P_{ij} is the probability that the Markov chain transition from state i to state j. These transition probabilities represented in an $N \times N$ matrix $P = (P_{ij})$ is the *transition matrix* of the Markov Chain. A transition matrix is a right stochastic matrix. That is, given

a sequence (e.g., state-trace data) with finite state space S with cardinality S, the transition matrix P can be represented as

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,j} & \dots & P_{1,S} \\ P_{2,1} & P_{2,2} & \dots & P_{2,j} & \dots & P_{2,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \dots & P_{i,j} & \dots & P_{i,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{S,1} & P_{S,2} & \dots & P_{S,j} & \dots & P_{S,S} \end{bmatrix}$$

$0 \leq P_{ij} \leq 1$, and $\forall i$ we have,

$$\sum_{k=1|\ k\in S}^{S} P_{ij} = \sum_{k=1}^{S} P\ (X_{n+1} = k | X_n = i) = 1 \tag{3.3}$$

Given a state i, the next state must be one of the possible states and the sum of all probability values in each row must be 1.

3.1.4 An Exploratory Case-study: M/M/1 Queueing System

Queueing theory is the mathematical study of formation, function, and congestion of waiting lines, or queues (Shortle et al., 2018). In queueing theory, a queueing model is constructed to design and evaluate the performance of a queueing system based on several measures, such as the server utilization, the length of waiting lines, and the delays – or waiting times – of entities. Discrete event simulations are heavily used in the analysis of queueing systems. This section introduces a simple single-server queueing system (see Figure 3.1); similar to the example that can be found in L'Ecuyer et al. (2003), Lang et al. (2015), and (Law, 2015).

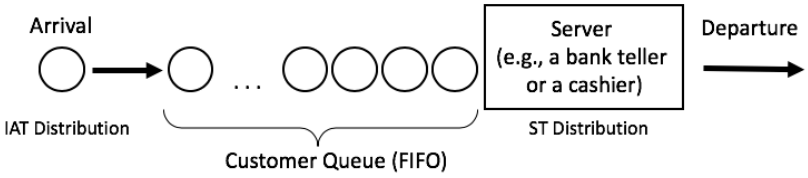

Figure 3.1. A single server queueing system; IAT stands for inter-arrival time; FIFO is the first-in-first-out queue ordering

Represented in Kendall's Notation (Kendall, 1953), an M/M/1 queue[10] is a specific case of an M/M/c multi-server queue, where the number of servers c = 1, the arrivals of the entities is determined by a Poisson process with rate λ (so the interarrival time IAT is Exponentially distributed with parameter 1/λ), and the service times are distributed according to an exponential distribution with rate μ (so having an average service time ST of 1/μ) (Bhat, 2015). The flowchart given in Figure 3.2 visually represents the sequence of steps that an entity, such as a customer takes. The entity interarrival times A_1, A_2, … and the service (or delay) times S_1, S_2, … of each successive entity are iid random variables. If a new entity arrives and the server is idle (i.e., the queue is *empty* and the queue length is 0), then the entity *seizes* the server and *delays* for the duration of service time. The server is *released* once the entity seizing it has been served (for the duration of the service time) and can be seized by the next entity in the queue in a first-in, first-out (FIFO) manner. If a new entity arrives and the server is in state *busy*, this new entity joins the end of the single queue. The server goes back to the state *idle* if the queue becomes *empty* (i.e., the queue length is 0) after completing a service for an entity. Note that the queue length is the number of entities in the queue waiting to be served and the capacity of the system at a given time is the queue length + 1 entity in the server.

A number of world views can be used to express a conceptual model (Fishman, 1973). In this case study, we modeled the above described single-server queueing system based on the event-scheduling world view (see §2.2.4.1). We used the open-source DSOL (Distributed Simulation Object Library) simulation engine to implement the queueing system and to generate the state traces for the model (Jacobs, 2005). Using commercial software was considered for this purpose, but rejected because it might be that certain internal state variables would be unknown or inaccessible to create a full state trace. An open-source simulation engine allows us to get access to the full state vector at any point in time, and to output the state at events or sample the full state at regular intervals.

[10] M stands for Markovian or memoryless.

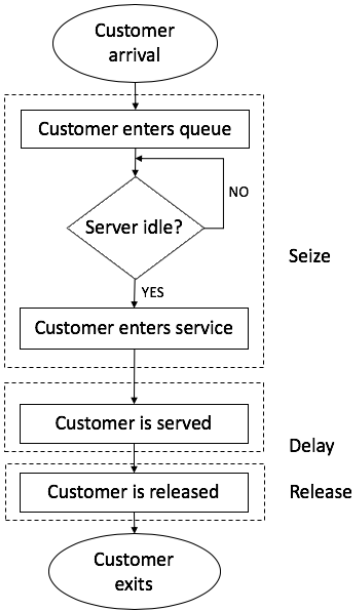

Figure 3.2. Flowchart of the M/M/1 queuing simulation (Jacobs & Verbraeck, 2006)

Although an M/M/1 queueing system is an over-simplification of the complex systems of real interest (i.e., large-scale complex systems), they are highly similar in terms of how our method is applied. Furthermore, the theoretical model of an M/M/1 queue is well-established in the literature. Therefore, the analytical results of our lumped model can be validated against an M/M/1 discrete event simulation model, and against the analytical values that are well known. In the following subsections, we will provide more information on our M/M/1 model specific implementation and the simulation details.

3.2 The Temporal Data Mining-based Method for DEVS Model Abstraction

In what follows, we present the steps of our temporal data mining-based method for discrete-event simulation model abstraction and address several considerations and actions for modelers. These can be grouped into four main categories:

(I) Considerations and actions regarding the generation of state-trace data (see §3.2.1)

(II) Considerations and actions regarding the application of the temporal data mining tasks to the state-trace data and the generation of the transition probability matrix *P* of the discrete-time Markov Chain (see §3.2.2)

(III) Considerations and actions regarding the generation and the simulation of the discrete-time Markov Chain (see §3.2.3)

(IV) Considerations and actions regarding the validation of the Markov Chain results (see §3.2.4)

3.2.1 Generation of Discrete-event Simulation Model State-Traces

The first step of our method is the generation of state-trace data from the execution of a base model. The role of state-trace data in our method is twofold: first, state-trace data are used to calculate the state transition probabilities. These probabilities represented in a (sparse) matrix format form the transition matrix *P* of the Markov Chain. Second, state-trace data can be used to validate the resulting Markov Chain model against the base model. For these two purposes, any generated model state-traces will have to be split into two subsets: (1) a *training* set that will be mined by the frequent episodes mining algorithm to detect frequent state transitions and construct a transition probability matrix for the Markov Chain, and (2) a *validation* set that will be used to compare outputs (e.g., key performance indicators [KPIs]) estimated by the Markov Chain to those generated from the original model in order to evaluate the accuracy of our model abstraction.

Before executing the model, the modeler needs to consider several factors. As will be shown in the next subsections, the following considerations are important: *representation of time* (see §3.2.1.1), *the type of model (stochastic vs. deterministic)* (see §3.2.1.2), *the type of simulation (terminating vs. non-terminating)* (see §3.2.1.3), *the inclusion of input and output variables* (see §3.2.1.4 and §3.2.1.5), and *the complete model state-trace vs. partial model state-trace* (see §3.2.1.6).

3.2.1.1 Representation of Time

One of the considerations of the modeler when generating state-trace data is the representation of time in the state-trace data. The final decision of the modeler should be guided by the following two questions:

1. How can the simulation time be embedded in the state-trace data?
2. In what format (elapsed, absolute, or fixed-rate) and with what type of variables (categorical, numerical, hybrid) should the time in the state-trace data be represented?

How the simulation time can be embedded in the state-trace data is dependent on the time advance mechanism *ta* of the discrete-event simulation (base model). When working with discrete event simulations, the two possible time advance mechanisms that can be implemented are the *next-event time advance (progression)* and *fixed-increment time advance* (Law, 2015):

(I) *Next-event time advance:* If next-event time advance mechanism is chosen as the time advance strategy, the internal clock of the simulation is advanced when an event occurs. Thus, state-trace records can be generated at the time of event occurrences and the simulation time can be stored as a variable (column) in the state-trace record. The type of non-uniform time sampling of state-trace data is useful to preserve the information on the event-time progression of the simulated system.

(II) *Fixed-increment time advance:* In discrete-time simulations (DTSS), there is a fixed time-step size Δt that is the uniform increment at which the simulation clock is advanced (Law, 2015). In discrete-event simulations, a fixed time-step mechanism can be implemented by scheduling dummy events at every Δt time unit (Law, 2015). This type of time advance mechanism allows modelers to generate a new state-trace record at fixed time intervals, and to store the simulation time as a part of the state-trace record. State-trace data sampled with equidistant time indexing have the characteristics of *time-series* data (see 2.3.1 for temporal data types). For this sampling strategy, Δt is the natural selector of the sampling rate. An increase in the size of the fixed time-step Δt (i.e., a decrease in the sampling rate) results in a lower level of quantization, hence, a reduction in the state-trace length (i.e., lesser state-trace records or rows). This reduction in the state-trace length can positively affect the run-time required for the frequent episode mining algorithm to complete its task. On the other hand, if

the elapsed times of the events in reality are always less than the Δt, the lumped model might not express the behavior of the base model with sufficient precision. Finding the optimal Δt can be a difficult and an application-specific problem, often requiring specific domain and model knowledge.

The considerations regarding the representation of time relate to the format (elapsed time or absolute simulation time) and the variable types (categorical, numerical, hybrid) of the time information. This decision may be guided by the dependency of the base model's state-transitions to the absolute time (e.g., systems that are busier on certain times of the day; or a service that is always triggered at 17:00; or a certain output that is triggered just before the end of the simulation run) or elapsed time (e.g., how much fuel has been added to a fuel tank given the pump is filling the tank with a constant rate) and by the limitations of the temporal data mining tool or the techniques (e.g., working with an algorithm that requires input that to be numerical). In some cases, the format dependency is on a derivative of the absolute simulation time, such as the time-of-day, the day of the week, the day in the year, or the month of the year. These values can be obtained with modulo functions from the absolute simulation time, and as such are easy to add to the state trace.

When the base model has a fixed-increment time advance mechanism, the modeler may exclude time entirely from the state-trace data. This is because the elapsed time or fixed sampling rate for such models would remain constant during the entire simulation run and, therefore, in the state-trace data. Furthermore, recording the absolute time for such models with a fixed-increment time advance mechanism would result in state-trace data with a time column with monotonously increasing values. Such representation of time in the data may negatively affect the performance of the frequent episode mining algorithm (we will elaborate more on the implications of variables with monotonously increasing or decreasing values on the performance of the mining algorithms in §3.2.2.1). In such cases, the time of the state changes can always be retrieved in later stages of the method using the time of the initial state and the sampling rate. Once the modeler chooses to capture the time information in the state-trace data, a new variable (column) can be added to the state-trace to capture the time in the chosen format and with the chosen data type.

3.2.1.1 Case Study Application

In order to practically generate the state-trace data from the simulation of the M/M/1 model, we implemented a TraceWriter class, which generates the state-trace data in a csv format in which all rows representing the state-trace records are homogenous, that is, each row has the same number of base model variable values (i.e., the values of state variables, input/output variables, time variable). Furthermore, the model variable values from all rows with the same index create a column and all values in a single column represent the same model variable. To demonstrate the differences of state-trace data generated from a model with "next-event" and "fixed-increment" time advance mechanisms, this section will provide examples of M/M/1 model traces sampled with both. However, the results evaluated in §3.2.4 are obtained from the execution of the M/M/1 model with the next-event time advance mechanism.

In Table 3.1, the first ten state-trace records from an M/M/1 state-trace data are shown. The example state-trace data are sampled from an M/M/1 model with a next-event time advance mechanism. The first and second columns have the states of the *Encoded_server_status* (idle = 1, busy = 2) and the *Encoded_queue_length* {10 = 0, 11 = 1, 12 = 2, 13 = 3, …, n+10 = n}[11].

Table 3.1. A sequence-based state-trace data set based on next-event time advance

M/M/1 simulation model state-trace (next-event based, no time information)	
Encoded_server_status	Encoded_queue_length
1	10
2	10
2	11
2	12
2	13
2	12
2	13
2	14
2	15

[11] The decision to have the coded values for these two variables in the example state-trace data is for the readability of the example, that is, each variable has a non-overlapping data range. Note that the coded queue-length values were not used in the estimation of the performance measure "average queue length" (see §3.2.4).

2	14
…	…

Table 3.2 illustrates a portion of a state-trace data containing elapsed time as an additional information next to the uncoded versions of state variables *Server_status* (idle = 0, busy = 1) and *Queue_Length* (0, 1, 2, 3, …, n). Alternatively, the state-trace data could be populated with the absolute time instead of the elapsed time. Because the state-trace data is generated from the M/M/1 model with the next-event time advance mechanism, the state-variable values at each row would be exactly the same.

Table 3.2. A next-event time advance mechanism-based state-trace data set with elapsed time

M/M/1 simulation model state-trace (next-event based, elapsed time)		
Server_status	Queue_Length	Elapsed_time
0	0	0.00
1	0	1.234
1	1	0.739
1	2	1.374
1	3	0.982
1	4	0.003
1	3	4.484
1	4	0.375
1	5	4.060
1	4	2.405
…	…	…

On the other hand, the state-trace data shown in Table 3.3 is sampled with an M/M/1 model using a fixed-time increment mechanism. The sampling rate – 10 time-unit – is included in the data as a separate (the third) column. It can be seen from the *Queue_Length* variable that generating state-trace data from the M/M/1 model using the fixed-time increment mechanism with a relatively larger sampling rate may not be ideal to capture every change in the queue length. On the other hand, a state-trace data generated with a much smaller sampling rate (e.g., 1 time-unit) may result in capturing the same state more than once in the data.

Table 3.3. A fixed-increment time advance mechanism-based state-trace data with absolute simulation time column (sampled with a rate of 10 time-unit)

M/M/1 simulation model state-trace (fixed-time increment based, absolute simulation time)		
Server_status	Queue_Length	Absolute_simulation_time
0	0	0
1	0	10
1	1	20
1	0	30
0	0	40
0	0	50
0	0	60
1	3	70
1	2	80
0	0	90
…	…	…

3.2.1.2 Stochastic vs. Deterministic Models

If a simulation model is deterministic (i.e., the model has no random variable), the behavior of the system over time for a given initial state and particular input set would trace the same state transition history and generate the exact same outcome. Therefore, for a deterministic model, a single run of the model would suffice to obtain a state-trace that is representative of the model's behavior under the selected initial state and the set of input parameters. However, large-scale complex models are typically stochastic as the large-scale complex systems being modeled contains inherent uncertain properties. These uncertainties are implemented in large-scale complex models as random variables, whose values follow a probabilistic distribution (Serfozo, 2009). To accurately estimate the associated variability, the modeler should use replication to obtain independent and identically distributed observations. In our method, one of the goals of the modeler is to

use replication to generate statistically independent (i.e., using different seeds values) and sufficiently long state-trace data from the base model (see §3.2.1.3 for more discussion).

Sufficiently long state-trace data are achieved by increasing the run length of the simulation, while statistical independence across runs is achieved by increasing the number of repetitions with different Random Number Generator (RNG) seed values (Law, 2015)[12]. Recall that we use state-trace data generated from the base model to identify frequent behavioral patterns (i.e., successive state-transition pairs) using a frequent episode mining algorithm and form the *transition matrix* of the Markov Chain. A Markov Chain's ability to correctly estimate the behavior of the original model depends on the quality of the transition probability matrix. Statistically independent and sufficiently long state-trace data generated with different seeds help estimate the probability mass function (for discrete random variables) more accurately. What constitutes a sufficiently long run length and number of repetitions depends on the model under study as well as the desired level of precision (i.e., margin of error). In the case study presented in this chapter, we will examine the performance of our method in terms of its ability to accurately estimate the original model's KPI's under a range of run lengths and repetitions.

3.2.1.2 Case Study Application

To determine the minimum number of repetitions and run length required to achieve this level of precision, we perform two sets of preliminary analyses. We measure the performance of the M/M/1 queueing system (with $\boldsymbol{\lambda} = 0.10$ and $\mu = 0.12$) based on the output estimates of three performance measures: server utilization $\boldsymbol{\rho}$, average waiting time w_Q, and average queue length L_Q. To assess the performance of our proposed method in terms of its ability to accurately estimate these three KPIs, we compare the output estimates obtained from the Markov Chain to those from the base model.

[12] Note that the individual data points (i.e., state-trace records) within a particular state-trace data generated from a simulation run are not iid. However, the different RNG seed value for each run ensures the independence across runs.

In the first set of preliminary analyses, we examine the effect of increasing the number of repetitions (while keeping the state-trace length of each independent run constant) on the Markov Chain's accuracy. Recall that we split the runs of the base model into two equal non-overlapping subsets for *training* and *validation* purposes. The total number of runs will therefore always be twice the number needed for the validation tests. We start with an experiment containing a total of 10 runs (20 in total for training and validation) of the base model and gradually increase the runs {20, 50, 100, 200, 500} to achieve the desired level of precision, i.e., a margin of error of the Markov Chain generated KPI of <1%. To ensure independence across runs, we assign a different seed value to each run. Note that the length of each state-trace data generated from each independent run for a fixed run-length is a random variable, depending on the observed values for the interarrival and service-time random variables. In order to have a fixed length of state-trace data from each independent run, the modeler should implement an additional limiting logic that either (a) stops the data collection when the state-trace data reach to a pre-determined length or (b) postprocess each state-traces to remove the records above the length threshold. In this set of analyses, each run has a run-length of one million-time units and once the data generation is finished, all the state-traces data collected from those runs are postprocessed to a fixed length of 50,000 state-trace records.

In the second set of preliminary analyses, we examine the effects of increasing the length of state-traces (while keeping the number of repetitions constant) on the performance and the precision of the Markov Chain. Similar to the first set of experiments, we gradually increase the state-trace lengths (1,000; 2,000; 5,000; 10,000; 20,000 and 50,000 state-trace records). In each experiment, the number of repetitions is set to a fixed value which is determined by the outcome of the first set of experiments (see §3.2.4 for more details).

For each individual experiment within the two sets of preliminary analyses, we calculate the performance indicators and their margins of error. By doing so, we will illustrate how much precision we will gain by increasing the run lengths and number of replications.

3.2.1.3 Terminating vs. Non-terminating Simulations

Shannon (1998) defines *simulation* as a twofold process: designing a model of a real-life system and conducting experiments with this model to gain knowledge about this system. Once the model has been developed, verified, and validated[13], the next steps for a simulation study are the *experimental design* and *experimentation* (i.e., the execution of the simulation experiment; Shannon, 1998). In the experimental design step, the goal is to design an experiment that will successfully provide answers to the questions about the system and to determine how each test run is executed. In the experimentation stage, the focus is on the execution of the simulation experiments and the generation of desired data and statistics for analysis. There are several considerations for the experimentation stage, including the decision on the length of a simulation run (i.e., sample size), the identification of the starting conditions of the model, and the decision to include or exclude a warm-up time. These considerations are largely determined by the type of the simulation, which can be *terminating* or *non-terminating*.

In a *terminating simulation*, the simulation starts at a defined initial state or time and ends when a defined terminating event is received or a time is reached. When experimenting with terminating simulations, the modeler must to decide on (a) the initial-state of the simulation, (b) the terminating condition which defines the run-length of a single simulation run, and (c) the number of replications (Hoad et al., 2010). The terminating condition influences the maximum length of the state-trace data that can be generated from a simulation run. Obtaining sufficiently large state-trace data may not always be possible due to the terminating condition. To achieve that, the modeler must often make many replications with the same initial distribution with each replication having a different Pseudo-RNG seed value (Kleijnen, 2017; Law, 2015).

In a *non-terminating simulation*, also referred to as *steady-state simulation*, there is no end-state or an end-time and the simulation could theoretically continue indefinitely (Shannon,

[13] Note that validation and sometimes verification are also special types of experiments with this model, from which we can also learn about the system.

1998). This gives the modeler the ability to choose between making a single long run and making many independent runs. Whitt (1991) shows that both options are usually as efficient as long as the independent replications are sufficiently long enough to obtain good enough estimates. However, it is still considered to be important to make independent runs with different RNG seed values to overcome a possible dependency on a selected seed.

Another important task of the modeler is to determine the warm-up period. The behavior of the transient period is different from the steady-state period in a non-terminating simulation. When studying the stochastic behavior of non-terminating systems, we are typically only interested in the steady-state behavior of the system in a *non-terminating simulation* (Kleijnen, 1984). However, there will be a difference between the estimator's expected value and the value it is estimating when there is no warm-up period to comfortably pass the transient period before collecting data for analysis or a realistic initial condition (Schruben et al., 1983; Whitt, 1991). This difference is known as the *initialization bias*, or *start-up problem* (Law, 2015). Several methods have been proposed to mitigate the initialization bias in steady state discrete event simulation (refer to Mahajan & Ingalls, 2004 for an array of references).

To reduce the effects of initialization bias in our proposed method, the modelers could choose (1) to collect state-trace data from the original model for both transient and steady-state period, use the data to generate two Markov Chains for both transient and steady-state period, but only perform data analysis for the steady-state period Markov Chain; or (2) not to simulate the transient period and, instead, introduce a single admissible initial state, which then becomes the initial state of the steady-state period Markov Chain. This initial state is used in all replications as the starting condition; thus, it is recorded as the first state-trace record in all the state-traces data. Although eliminating the simulation of the transient period might bias the final estimator, we can assume that the effect of the bias becomes negligible with large sample size (Kleijnen, 1984).

3.2.1.3 Case Study Application

The type of simulation implemented for the M/M/1 case study is a non-terminating simulation. There is no external event that determines the 'ending' of the simulation and state-trace records are generated at each event occurrences until the simulation is halted by scheduling a special "terminating" event at time unit 1,000,000-. We determined that this run length is sufficient to generate at least 50,000 state-trace records, which is determined as the sufficient data length for the desired level of precision; margin of error of <1% (see §3.2.4 for more details on the calculation of margin of error for the M/M/1 case study).

Furthermore, we concluded that each independent run – out of 50 repetitions – will begin with the initial state P_0: *empty-and-idle* and we do not use a warm-up period. The reason to exclude the warm-up period from the simulation experiment is twofold: First, the initial state P_0 is realistic and the system has a reasonable chance to be in this particular state. Second, there are no entities in the system that would not have a 'history' for statistics calculations when we start in the *empty-and-idle* state.

We will demonstrate the process of generating state-trace data from terminating simulations using a battlefield case study in Chapter 4.

3.2.1.4 Inclusion of Input Data

When experimenting with a simulation model, one of the goals is to understand the effects of changes in the input data to the output of the simulation, such as the performance measures (Maria, 1997). Unless the simulation model being used does not receive any external input and the simulation experiment is designed to have all runs with the same set of input, the desired analysis for the simulation study may require running the simulation model with different set of inputs and these different set of input variable values may need to be included in the state-trace data. In our method, the inclusion of input variable values in the state-trace data for a particular run is done by adding the variables as columns in the data.

The state-transitions simulated by the state-walk of a Markov Chain are dependent on the input set (in addition to the RNG seed value used as explained earlier in §3.2.1.2). In other words, the transition probabilities in a transition matrix *P* calculated from a collection of state-traces in a training set generated from the base model using a specific set of input values can be different from state-traces in the same training set generated using a different set of input values. This is also true for the traces in the validation set, which will be used to validate the output of the Markov Chain. Therefore, the further separation of the validation and training data into subgroups per input set provides modeler the ability to simulate the Markov Chain using a transition matrix calculated from *a single input data-origin training data* (i.e., the collection of state-traces generated from the simulation of the base model with the same input data and reserved as training data) and validate the output of the Markov Chain with the corresponding *same single input data-origin validation data* (i.e., the collection of state-traces generated from the simulation of the base model with the same input data as the training data, but reserved as validation data).

The subcategorization of state-trace data into training and validation sets per input data may be done manually by organizing the training and validation sets based after their generation, or can be automated by introducing an additional identifier variable included in the state-trace data as a separate column (in addition to the input variables) so that our method may later identify the input set used for each run. Note that the identifier column value will remain constant throughout a run and each state-trace data generated from the execution of the base model with the same input set contains the same identifier value. When the base model is simulated with a new set of inputs, then a new identifier value is generated (e.g., by using an integer variable as the identifier and incrementing the value by 1) and a lookup text file that contains the map of identifiers and their corresponding input set is updated. Note that the necessity to separate state-trace data set into subcategories of validation and training sets per input data comes with the necessity to introduce an additional logic to choose the best fitting Markov Chain when the modeler wishes to simulate the Markov Chain with an input data that was not among the set of input data previously used to execute the base model while generating the state-traces. An approach might be that when quantizing the input variables, the modeler could use the quantized value and trained the model for all combinations of all quantized values. When linearity

can be assumed, the modeler may run the model with the value above and below, and then interpolate the output values from both (sets) of runs with the Markov model.

3.2.1.4 Case Study Application

The M/M/1 model we use for our case study has a single set of input data. The input variables and the values of the M/M/1 model are given in Table 3.4 below:

Table 3.4. The input variables and the values of the M/M/1 model

Input variable	Value
Average arrival rate λ	0.10
Average rate of service μ	0.12
Number of servers c	1

Because the model has a fixed (single) set of input values and there is no external input, the inclusion of the identifier variable and the input values to state-trace data as columns, which remain constant for all state-traces, will not affect the variability of the episodes discovered by the EMMA algorithm. Therefore, we exclude the input data from the state-traces generated from the execution of the M/M/1 model.

3.2.1.5 Inclusion of Output Data

Depending on both the preferences of the modeler and the goals, requirements and the type of analysis needed for the simulation study, output variables can be included in the state-trace data. In a simulation model, the modeler can calculate and report different types of run statistics (e.g., tally, time-persistent) to estimate the output performance of the model. A *tally* or *tally statistic* computes and represents the min, max, mean, variance and standard deviation values of the observations. *Time-persistent statistics* are time average statistics and often used for computing the time averages or time-dependent performance indicators. In our method, one way to include the output data of a simulation

model in the state-trace data is to directly incorporate the transient values of these run statistics as columns.

On the other hand, the modeler may prefer to include the raw state variable values of the base model (e.g., the queue length and duration) in the state-trace data to instead of the run statistics. Note that the decision to add run statistics or individual variable values of the base model when generating the state-trace data will also determine the content of the state-trace data reproduced by the Markov Chain. Therefore, although the strategy to include individual variable values requires the modeler to (a) ensure that all state-variables needed for the run statistics calculations are included in the state-trace and to (b) implement the code needed to perform the calculation of the run-statistics as an additional step, the extra information in the state-trace data can be useful for the debugging of the Markov Chain model.

3.2.1.5 Case Study Application

Based on the input values presented in §3.2.1.4, the theoretical calculations of several mean measures of the M/M/1 model are given below:

In this M/M/1 case study, we are particularly interested in the estimation of the following three KPIs: server utilization $\boldsymbol{\rho}$, average waiting time w_Q and average queue length L_Q. The DSOL implementation of the M/M/1 base model does the capturing of these three KPIs as shown in Figure 3.3. The lines 373-376 are the run statistics. In line 374, the SimTally class is instantiated and the tally object (dN) is subscribed to *Seize.DELAY_TIME* events fired by the queue. The line 375 defines a sim-persistent statistic (qN) that is subscribed to *Seize.QUEUE_LENGTH_EVENT* events fired by the queue. Finally, the line 376 introduces a Utilization object computing and representing the utilization of a station (i.e., the server), which is also a time-persistent statistic. To calculate the mean of these three performance measures, we call the *getSampleMean*() method at the end of each run.

```
365             // The flow
366             this.generator.setDestination(this.seize);
367             this.seize.setDestination(this.server);
368             this.server.setDestination(this.release);
369
370             // Statistics
371             try
372             {
373                 this.seize.addListener(this, Seize.DELAY_TIME);
374                 this.dN = new SimTally<>("d(n)", getSimulator(), this.seize, Seize.DELAY_TIME);
375                 this.qN = new SimPersistent<>("q(n)", getSimulator(), this.seize, Seize.QUEUE_LENGTH_EVENT);
376                 this.uN = new Utilization<>("u(n)", getSimulator(), this.server);
377             }
378             catch (RemoteException rme)
379             {
380                 throw new SimRuntimeException(rme);
381             }
```

Figure 3.3. Model performance metrics directly embedded in the state-trace data

For this case study, we choose to individually trace the values of several key variables and include them as individual columns in the state-trace data, instead of including the run statistics directly. Table 3.5 shows an example of a portion of a state-trace data containing these key individual variable values.

Table 3.5. A portion of a raw-state trace data with the final data columns

Server_status	Queue_length	Elapsed_time	Time_in_queue	Num_of_observations
0	0	0.000	-1.000[14]	0
1	0	1.235	0.000	1
1	1	0.739	-1.000	0
1	0	1.374	1.374	1
1	1	0.982	-1.000	0
1	0	0.003	0.003	1
0	0	4.484	-1.000	0
1	0	0.375	0.000	1
1	1	4.060	-1.000	0
1	2	2.405	-1.000	0
…	…	…	…	…
1	4	0.837	-1.000	0

[14] The state -1.000 for the *Time_in_queue* variable means that there is no entity waiting in the queue.

Recall from earlier that the inclusion of the individual state-variables instead of the run-statistics means that we need to calculate each KPIs using the state-variable values obtained from the Markov Chain-generated state-traces. Below table shows the equations we use for the calculation of each KPI (left-side of the table) and the steps to be taken to calculate these KPIs using the state-trace data.

Table 3.6. Calculation details of the three KPIs

$\rho = \frac{\int_0^T \rho_t \, dt}{T}$	To calculate the average server utilization: for each interval (between events, or between observations), determine the utilization of the server $\rho_i \in \{0, 1\}$. Multiply by time interval τ_i , sum the values and divide the sum by total simulation time T (the sum of all interval lengths).
$\rho = \frac{\sum_{i=1}^{n} \tau_i \, \rho_i}{T} \; where \; T = \sum_{i=1}^{n} \tau_i$	For event based, all τ values are different; for observation based, they are the same.
$L_Q = \frac{\int_0^T L_{Q,t} \, dt}{T}$	To calculate the average queue length: for each interval (between events, or between observations), determine the queue length of the server $L_{Q,i} \in \{0, 20\}$. Multiply by time interval τ_i , sum the values and divide the sum by total simulation time T (the sum of all interval lengths).
$L_Q = \frac{\sum_{i=1}^{n} \tau_i \, L_{Q,i}}{T} \; where \; T = \sum_{i=1}^{n} \tau_i$	
$w_Q = \frac{\sum_{i=1}^{N} w_{Q,i}}{N}$	To calculate the average waiting time in the queue, sum the individual waiting times in the queue and divide by total number of entities N that have left the queue. This is a bit less straightforward to calculate than the other two KPIs (see the section below).

Note that not all events we register in the state-trace are events where an entity leaves the queue. This means that we have to indicate non-leaving events in the state-trace data for our method to identify. In our case-study model, we coded the non-leaving events as -1. When we have a total of E events, we calculate the average waiting time as follows:

$$w_Q = \frac{\sum_{e=1}^{E} w_{Q,e} \mid w_{Q,e} \geq 0}{\sum_{e=1}^{E} 1 \mid w_{Q,e} \geq 0}$$

In other words, we only sum the registered waiting times for values that are not -1, and divide by the count of waiting times that are not -1. Unfortunately, the above formula is a problem when the system is studied using observations rather than events, since multiple entities can leave the queue in one interval. Therefore, we implemented the following:

- store the sum (or average) of the waiting times $w_{S,i}$ for all entities that left the queue during interval i, and a value of *NaN* or -1 in case no entities left the queue during the interval;
- store the number of entities n_i that left the queue during interval i, where the intervals are numbered from 1 to N.

We then calculate the average waiting time as follows:

$$w_Q = \frac{\sum_{i=1}^{N} w_{S,i} \mid w_{S,i} \geq 0}{\sum_{i=1}^{N} n_i \mid w_{S,i} \geq 0}$$

Meaning that, we only sum the registered waiting times for values that are not -1, and divide by the total count of waiting times that are not -1, where the count is increased by 2 or more in case multiple entities left in the same interval.

3.2.1.6 Complete vs. Partial Model State-Trace Data

If a source system is defined as a set of input and output variables and a collection of observable and unobservable states (Klir, 1985), the effort of modeling large-scale complex simulation models is to specify a generative or a structure system with a number of states selected from the source system including the input and output variables; see Figure 3.4.a. Some of these states will be directly observable, but the power of using simulation modeling is that we can also infer the values of the states that are not directly observable, and include these state variables in our model as well. Therefore, another important consideration of the modeler in our method is the decision to generate a complete-model state-trace using all state variables of the original large-scale complex simulation model, or a partial-model state-trace data set by making a sub-selection of the states in the model; see Figure 3.4.b. Generating a partial-model state-trace data can be seen as a similar process to the iterative process of modeling a large-scale complex simulation model from a source system. The decision of a complete or a partial model state-trace is driven by the goals, requirements and limitations defined by the stakeholders (i.e., modeler, model users) and resources (i.e., computational resources, time).

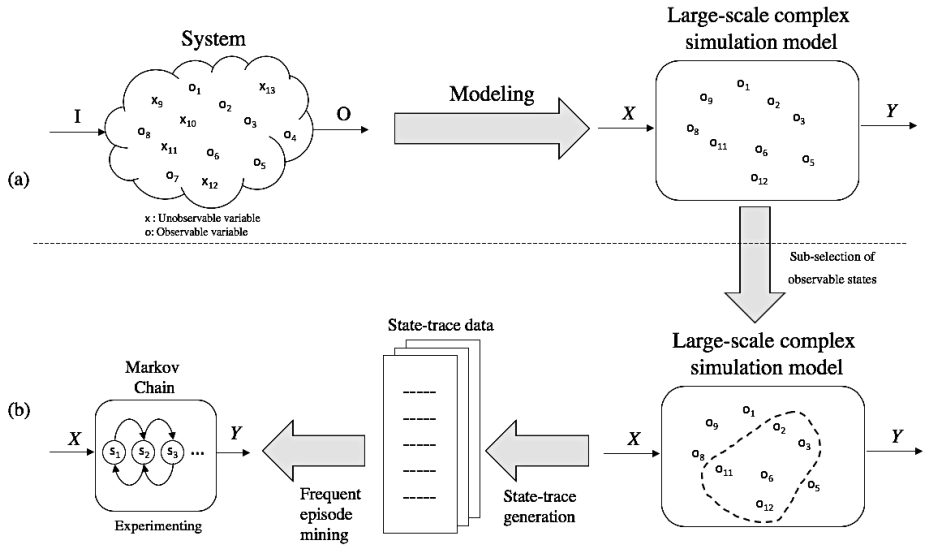

Figure 3.4. From System to Markov Chain progression

A state-trace is considered as a *complete-model state-trace* if it contains the complete set of variables that describes the state of all individual atomic components and, as a collection, the state of the system at a particular instant. Meanwhile, it is a *partial-model state-trace* when:

- the state-trace data have *a (selected) subset* of state variables of all atomic components of the base model, or
- the state-trace data have the complete set of state variables of *a (selected) subset* of atomic components of the base model, or
- the state-trace data have *a (selected) subset* of state variables of *a (selected) subset* of atomic components of the base model.

Although generating a complete-model state-trace is a more straightforward process, the size and variety of data may make working with the state-trace of large-scale complex discrete-event simulations impractical. On the other hand, the process of generating a partial state-trace data set is not a straightforward task that requires modelers to correctly identify subset of atomic components and the subset of state variables to be included in the state-trace data in order to generate valid Markov Chains.

3.2.1.6 Case Study Application

The M/M/1 model is a simple model with only 2 state variables. Therefore, the state-trace data generated from the simulation of the M/M/1 model and used in the temporal data mining step are complete-model traces.

We will demonstrate the process of making selections out of a large sample of variables with many intricate relationships with larger models in Chapters 4 and 5.

3.2.2 Application of the Temporal Data Mining Tasks to the State-Trace Data

The next step of our method is the application of frequent episode mining to the training data to construct a transition matrix for the discrete-time first-order Markov Chain. In

Appendix A.1, we explained why the EMMA algorithm is the best suited for the frequent episode mining task for our proposed method. In the following sections, we describe the considerations and actions regarding the *preprocessing of the state-trace data*, the *application of the EMMA algorithm* to identify all frequent episodes (i.e., all the possible state-transition pairs) from the preprocessed state-trace data and calculate their occurrence frequencies, and finally *the construction of the transition matrix* using the transition probabilities for all state-trace record pairs.

3.2.2.1 Preprocessing of the State-Trace Data

Data preprocessing involves the tasks of data cleaning (i.e., detecting and correcting/removing erroneous data and outliers, dealing with missing data), data integration (i.e., combining data from multiple sources into one), and data transformation (i.e., conversion, recoding, quantization, approximation or normalization of data values; (i.e., conversion, recoding, quantization, approximation or normalization of data values; Mörchen, 2006b). Preprocessing tasks can be applied to a *single state variable*, to *multiple state variables measured in the same units*, to a *single or multiple state-trace records*, and to a *complete state-trace data set*. This is a highly application dependent step; that is, the selection of type of preprocessing tasks and the particular algorithms to perform these tasks are determined by several factors, including the input file format prescribed by the chosen temporal data mining algorithm, the preferences of the modeler, and the specific goals and requirements of the model abstraction study.

In our proposed method, the type of preprocessing techniques to be used are determined by the input file format of the EMMA algorithm, as well as the preferences and the requirements of the modeler for the model abstraction task. The SPMF implementation (Fournier-Viger et al., 2014; 2016) of the EMMA algorithm is designed to find episodes with at least *minsup* times occurrence from a given event sequence. The input of the algorithm is a text file containing an event sequence, in which each row is an item set and an optional the timestamp of the item set, separated by the character "|". If the event sequence has no timestamp, the corresponding Boolean parameter "self_increment" can be set to *true*. In that case, the timestamp information of each row is processed as a

sequence number incrementing by 1. This is useful when the input data is a state-trace data, which is sequential and the time is embedded in the item set as a separate entry.

The SPMF implementation of the EMMA algorithm specifies that items in an item set are separated by a single space and represented by positive integers. Therefore, any nonpositive noninteger value (e.g., negative floating point, negative integer, non-numerical data) in the state-trace data belonging to a variable should be recoded to a positive integer. It is important that the recoding method applied to a particular variable should be applied to all variables measuring the same unit (e.g., all variables that measure temperature). Another approach to deal with the issue of having nonpositive noninteger values in the state-trace data is to encode every unique row (state-trace record) to a nonnegative integer value. This way, the data can be represented in the format usable by the EMMA algorithm and the encoded values can be decoded back to their original value format after the episode mining step. The dimensionality of the state-trace data, and therefore, the size will be reduced as a result of encoding state-trace records with a large number of data columns to a single positive integer value. Note that in case the modeler chooses the encoding strategy to deal with nonpositive noninteger values in the state-trace data, the quantization of continuous data should be done before the encoding of the state-trace data.

In a state-trace data set, a state variable or a system state can have categorical (binary, nominal or ordinal) or numerical values (continuous or discrete). Although many real-world phenomena are represented as continuous (e.g., speed, temperature, distance), continuous variables, also referred as continuous features (Dougherty et al., 1995), possess several challenges when mining frequent episodes from the input data. One reason is that floating point variables, by nature, consist of an infinite set of values. It can be the case that the same value instances never appear twice in the state set, while the density of certain value ranges can differ tremendously. This is also true stochastic values, where the densities of the values are important, but each run would generate different actual floating-point values, according to these densities. Another reason is that some continuous variables take values that are monotonously increasing or decreasing by design and no values will repeat itself during a run (e.g., distance covered by a driving vehicle during the time of observation). Inclusion of such variables with as many different numbers of

states as the number of state-trace records in the data (each value occurring only once) will undermine the ability of the frequent episode mining algorithms. The above points lead to the necessity to account for value ranges rather than points for non-categorial variables, as well as categorial variables with a large number of categories.

Quantization[15] is the process of converting or partitioning continuous data into a smaller, finite number of discrete values or categories. There are numerous quantization methods described in the literature. Liu et al. (2002) propose a hierarchical framework for quantization that guides the selection of the most suitable technique within overarching splitting and merging approaches, and supervised and unsupervised methods. A commonly used quantization method is the *binning*. The goal of binning is to quantize non-categorical variables or categorical variables with a large number of categories into a user-specified number of bins (Liu et al., 2002). This can be done by either dividing the value range of a non-categorical variable into a given k number of intervals (bins) with equal coverage of value ranges, also known as *equal-width binning*, or by placing equal number values in k number of bins, known as *equal-frequency binning*. Equal-frequency bins can be created using *quantiles*, i.e., values that split data into equal intervals (Field, 2013). Commonly used quantiles, or *q-quantiles*, are 2-quantiles or *median*, 4-quantiles or *quartiles*, 10-quantiles or *deciles*, and 100-quantiles or *percentiles*.

In the following case study application section, we will demonstrate the quantization of variables using various binning strategies.

3.2.2.1 Case Study Application

Following the completion of the considerations listed in §3.2.1 and generating the final version of the state-trace data (see Table 3.5), the next step of our proposed method is to identify the variables that require preprocessing. In this section, we will demonstrate

[15] Although data mining and machine learning literature use the term *discretization* and *quantization* interchangeably, we reserve the term discretization to refer only to the transformation of the continuous time variable to a discrete one, whereas quantization is aimed at the transformation of the continuous state variables.

how we applied quantization, specifically binning-based methods, to the values of three variables in the state-trace data: *Queue_length*, *Elapsed_time*, and *Time_in_queue*.

For the quantization of these three variables, we chose *binning* as the strategy. In order to choose the suitable binning strategy for each variable, we investigated the type of numerical values these variables take and the statistical distribution of these values. In the *Elapsed_time and Time_in_queue* case, variables take floating point numbers and have right-skewed (positive skewness) distributions; that is, the peak of the histogram is on the left side and it has a long right tail. The drawback of using *equal-width binning* with these types of variables is that we can end up splitting the data into irregular bins with some bins containing very small numbers of values. Therefore, we chose a quantile-based binning algorithm by (Rawashdeh, 2020) to quantize *both Elapsed_time and Time_in_queue variables*. Among the most commonly used q-quantiles, we picked *deciles* over *quartiles* to have a high enough number of bins for sufficient precision, and we picked *deciles* over *percentiles* to not undermine the ability of the frequent episode mining algorithm EMMA due to over population of bins. Hence, both variables were quantized in 10 bins, each containing 10% of the *Elapsed_time* and *Time_in_queue* scores. Note that it is important that the modeler omits any non-numeric or non-meaningful values from the input before quantizing the variable. In this example, when the queue is empty and therefore there is no waiting time, -1.000 is used as a fixed value for the *Time_in_queue* variable.

For the quantization of *Queue_length* variable, we used a different binning strategy. Unlike the *Elapsed_time* and *Time_in_queue* variables, *Queue_length* is an integer variable where the frequencies are large for values that are close to zero; e.g., queue length 0 has a much higher frequency than 1, and 1 is more frequent than 2, etc. In such cases, quantile-based or equal frequency binning is not ideal as the high frequency values would occupy more than one bin. For that reason, we use a custom strategy to quantize this variable, in which we set a minimum threshold of 5% of scores per bin. That is, frequent values (>5% of scores) are assigned their own bin, and infrequent values (<5% of scores) are grouped together until each bin contains at least 5% of the data.

In Chapter 4 and 5, we will demonstrate the effect of different levels of quantization to the performance and the accuracy of a Markov Chain using simulation models with relatively larger number of model components and a larger state space than the M/M/1 model.

3.2.2.2 Application of the EMMA Algorithm to Preprocessed State-Trace Data

After the preprocessing of the base model's state-traces, the next step in our method is to apply the EMMA algorithm to the preprocessed training data set to obtain state-transition probabilities and construct the corresponding Markov Chain transition matrix. In §3.1.3, we established that the memoryless property of Markov Chain states is akin to the discrete-event model states, and the state-transition probabilities represented in matrix format is the transition matrix of a valid stochastic discrete-time Markov Chain. Therefore, we set the *maxwin* input parameter to 2 and the *minsup* input parameter to 1 (see Appendix A.2 for the complete EMMA algorithm input parameter list) for the EMMA algorithm to discover: (a) all unique episodes with length 1 (i.e., unique state-trace records or unique states) appearing in the state-trace data at least one time, and (b) all unique episodes with length 2 (i.e., successive state-trace record pairs, or state transitions) with a minimum frequency of 1.

A Markov Chain simulation (see §3.2.3) requires three main elements: the state space S, an initial distribution vector $\boldsymbol{\pi_0}$, and a transition matrix $P = (P_{ij})$. The Markov Chain's state space S of the Markov Chain is equivalent to the set of all unique states discovered by the EMMA algorithm. Similarly, using the frequency information obtained during the episode mining stage, we can construct a transition matrix P of the discrete-time Markov Chain using the state-transition probabilities $P(ij)$ obtained from the full training dataset. Although a transition matrix of a Markov Chain is an $N \times N$ square matrix, Reichel et al. (2015) argue that when implementing Markov Chains, it is often computationally more advantageous to represent transition matrices as sparse matrices due to the high number of transitions that are considered impossible for the model to reach. In the following case study application section, we will show the sparsity of Markov Chains' transition matrices for the M/M/1 model.

3.2.2.2 Case Study Application

To perform the experiments required for preliminary analysis in the case study application section of §3.2.4, we designed two preliminary analysis data sets containing twelve transition matrices and their corresponding state sets (i.e., set of all unique states):

- *Preliminary analysis data set1* consists of state-trace data with increasing number of repetitions (10, 20, 50, 100, 200, 500) and fixed state-trace length of 50,000 state-trace records per simulation run
- *Preliminary analysis data set2* consists of state-trace data with increasing length (1,000; 2,000; 5,000; 10,000; 20,000 and 50,000) and fixed number of 100 repetitions

The Table 3.7 shows the effect of increasing number of repetition and trace length to the number of unique states (N) identified by the EMMA algorithm (for the given parameters of *minsup* =1 and *maxwin* = 2). Even though the data points are limited, the results suggest that with an increase in sample size (e.g., due to the increased number of repetitions or increased trace lengths) the number of unique states discovered by the EMMA algorithm increases. Given the number of unique states, the size of the resulting right stochastic transition matrices can be calculated as ($N \times N$).

Table 3.7. Transition matrix dimensions and the number of unique states discovered per data set

Preliminary analysis data set 1 *(fixed length of 50,000 state-trace records)*		Preliminary analysis data set 2 *(fixed number of 100 repetitions)*	
Repetitions	Unique states (N)	Trace length	Unique states (N)
10	966	1,000	892
20	991	2,000	930
50	1,012	5,000	961
100	1,029	10,000	985
200	1,046	20,000	1,004
500	1,063	50,000	1,029

The experiments also revealed that the generated transition matrixes from the experiments are sparse (i.e., containing many zeros) because (a) we are only interested in the

transition probabilities of successive states, and (b) the state-trace data generated from the M/M/1 model with the next-event time advance mechanism does not contain self-transitioning states, resulting in the probability values of zero in the diagonals of the transition matrices. For instance, the $N \times N$ transition matrix generated from the data with 10 repetitions and 50,000 state-trace records (per simulation run) contains 35,873 non-zero probabilities out of 933,156 (i.e., 966×966) probability values in an $N \times N$ transition matrix, which is only 3.84%. Similarly, only 5.43% of the 1,129,969 (1,063×1,063) probability values are non-zero in the data with 500 repetitions and 50,000 state-trace records (per simulation run). Figure 3.5 depicts a portion of the 966×966 transition matrix's heatmap, which only contains state-transitions between the first 50 states. Higher transition probabilities are represented in the heatmap by darker shades. The figure shows that many transition probabilities are 0 (in white color), as expected.

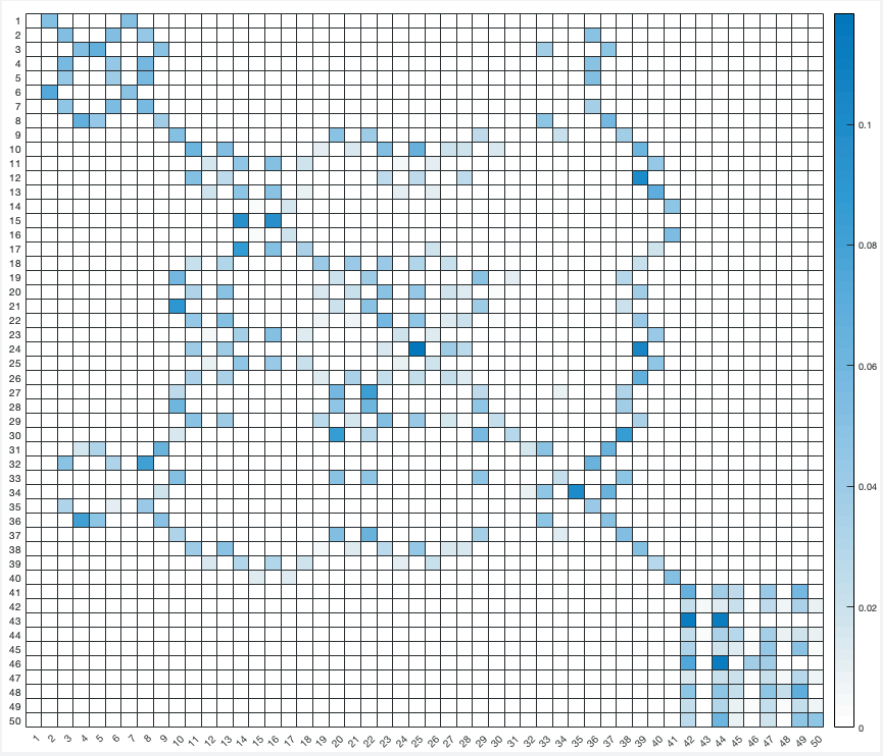

Figure 3.5. A portion of the 966×966 transition matrix's heatmap containing only the transitions between the first 50 states

3.2.3 Simulation of the Discrete-time Markov Chain

Following the generation of a discrete-time Markov Chain's transition matrix P, the next step in our method is to simulate the discrete-time Markov Chain to generate state-trace data for the final validation step. A Markov Chain generates a stochastic path by performing state walks when simulated. Appendix B contains an implementation of a generic state walk algorithm (see Table B.3) and more details on the implementation of the Markov Chain implementation used in the case studies. If the base simulation is non-terminating, the modeler should terminate the data generation from the Markov Chain once the desired number of state-trace records have been generated (for more information, see §3.2.4, "Preliminary analyses: Selection of minimum state-trace length"). On the other hand, if the base model is terminating, the Markov Chain simulation should automatically terminate once it reaches its end-state after *n* consecutive independent trials, or state walks. The geometric distribution represents a Markov Chain's probability of getting the first occurrence of a particular state, such as the end-state (see §5.2.4 for a more in-depth discussion on the geometric distribution).

The number of runs required to generate sufficient state-trace data from a Markov Chain simulation is determined by the number of runs used in previous experiments with the base model to generate the validation set. The modeler should use different random number generator seed values for each run to obtain iid data.

3.2.3 Case Study Application

As explained in the previous case study application section and further described in §3.2.4, we designed twelve preliminary experiments with varying run lengths and repetitions. Because the M/M/1 simulation is non-terminating, we set the artificial trace-length, which is the run-length (e.g., 1,000; 2,000; 5,000; 10,000; 20,000; and 50,000) required by the particular experiment when simulating the Markov Chains. MATLAB is used to implement the Markov Chain that was used in the simulation experiments for the M/M/1 case study. Appendix B.2 contains details of the MATLAB implementation.

3.2.4 Validation of the Markov Chain Results

An important step in any simulation study is to ensure the simulation model's validity by investigating and confirming that the simulated model is a sufficiently accurate representation of the system of interest (Robinson, 1997). Within the context of our proposed method, it is to test whether our model-of-a-model represents the original model accurately enough for the purpose we have in mind. Once a simulation model is implemented and simulation experiments are performed to generate data, the next step for the modeler is to ensure the simulation model's operational validity. Operational (results) validation is the process of determining to what extent the model-generated output behavior is similar to either the system output behavior or another model output behavior (Sargent, 2013). Therefore, the fourth step of our method is to use operational validation techniques to compare the Markov Chain generated data to the validation set.

3.2.4.1 Comparison of Estiamates Using Statistical Techniques and Criteria

Various operational validation approaches and techniques are described in the literature (see Balci, 1994; Roungas et al., 2018; Sargent, 2013 for a list of references), which range from more objective or mathematical/statistical procedures (e.g., hypothesis tests or confidence intervals) to more subjective approaches. Generally, a combination of techniques is used to evaluate the simulation model's validity. Within the statistical approaches, hypothesis tests can be used to objectively compare statistical properties (e.g., frequencies, means, distributions, variance) of the simulation model and the original system or another model to determine whether the simulation model's output has an acceptable accuracy for its intended usage. In our method, such statistical properties including model related performance measures can be obtained from state-traces. For instance, frequency distributions of state-transitions for the original model and from the Markov Chain model can be compared using the chi-square (χ^2) test (Anderson & Goodman, 1957). In this case, the modeler would test the null-hypothesis that there is no difference between the state-transition frequency distributions from the Markov state-traces and the original model generated test set. The chi-square test (Pearson, 1900; Plackett, 1983) uses a cross-tabulation to present the two distributions and to compare how well the observed cell counts (i.e., frequencies) fit with the cell counts that would be expected if the

distributions were equal to each other (i.e., if the null hypothesis were true). Similarly, model related performance measures of the original model and the Markov Chain model can be compared using the Student's *t*-test to assess how different the mean values on these measures are between the two models. The *t*-score is a ratio between the difference *between* two groups (i.e., validation set generated from the original model and the data generated from the Markov Chain) and the difference *within* the groups. The larger the *t*-score, the more difference there is between groups. The smaller the *t*-score, the more similarity there is between groups, in our case the model outputs. Regardless of the statistical technique chosen, the calculated test statistic (e.g. chi-square statistic or *t*-score) and degrees of freedom from the test are then compared to a critical value to calculate the *p*-value. If the *p*-value is larger than the alpha level chosen (e.g., .05 or 5%), any observed difference is assumed to be explained by sampling variability rather than a true difference. On the other hand, if the *p*-value is smaller than the chosen alpha level, the probability of finding the given test value due to chance or random sampling variability if the null-hypothesis were true is considered so small that we reject the null-hypothesis.

However, it is important to note that significance testing based on the *p*-value is sensitive to sample size (Sullivan & Feinn, 2012). With a large enough sample, unless the difference between the two distributions is very close to zero, a statistical test will almost always demonstrate a significant difference, even when the alpha is set at a more conservative level (e.g., .01 or .001), a phenomenon sometimes labeled "the large sample size fallacy" (Lantz, 2013). Yet, significant differences are in such cases often extremely small and not necessarily meaningful. For this reason, it is critical to not solely rely on the *p*-value (Lin et al., 2013), but also evaluate the effect size, which describes the magnitude of the difference (Sullivan & Feinn, 2012). For the chi-square test, one can use the Cramer's V measure for effect size (Sullivan & Feinn, 2012).

An alternative approach to the statistical hypothesis testing, or an additional approach in some cases, is to compare two probability distributions using the non-parametric two-sample Kolmogorov Smirnov test ("Kolmogorov–Smirnov Test," 2008) or to use other measures such as distance or divergence (see Cha & Srihari, 2002 and references therin). In our method, the two probability distributions (i.e., the probability distribution of the test set of the original model and the Markov Chain) are discrete probability distributions.

Therefore, it is important that the similarity or dissimilarity measures selected by the modeler should be applicable to discrete distributions, such as Bhattacharyya distance (Kailath, 1967) or the Earth Mover's distance (Rubner et al., 2000) as the statistical distance measures, or Kullback-Leibler divergence (Kullback & Leibler, 1951) as a divergence measure to calculate the similarity/dissimilarity between two datasets.

Although the above methods of statistical inference and distributional similarity metrics are preferable for evaluating operational validity as they allow for decisions on the basis of established and commonly used criteria, in practice it may not always be feasible to use these approaches (e.g., due to limitations of the data or the sensitivity of statistical tests to sample sizes limiting the interpretation of results) to determine whether the model (or the model of a model) is a satisfactory valid model for the model's intended purpose or application. In such cases, visualization of the simulation output can provide an additional perspective to understand the output visually and intuitively (Lin et al., 2013; Sargent, 2013). Within the context of our research, the modeler can compare the output estimations of the performance measures obtained from the validation set and the Markov Chain side-by-side by plotting the confidence intervals, or compare the frequency distributions using a histogram.

Where possible, the modeler should combine the outcomes of multiple measures to judge the operational validity.

3.2.4 Case Study Application

At the validation step, our main goal is to compare the state-trace data generated by the Markov Chain and the validation set obtained from the original M/M/1 model. Before we statistically compare the output estimates from both models, recall from §3.2.1.2 that we wish to obtain estimates with a precision of <1% margin of error. To achieve this level of precision, we need to determine the minimum number of repetitions and state-trace length. We run our Markov Chain with increasing numbers of repetitions (first set of preliminary analyses) and increasing length of state-trace data (second set of preliminary analyses), and evaluate the margin of error obtained in each experiment.

Preliminary analyses: Selection of minimum number of repetitions

Table 3.8 shows the mean performance measures and their margins of errors [MOE] for the first set of preliminary analyses. As becomes clear from this table, with larger numbers of repetitions, the margin of error becomes smaller and the level of precision of our estimate increases. For the KPI server utilization, a high level of precision (i.e., below 1%) is already achieved with as few as 10 repetitions. For the average waiting time and average queue length measures, the margin of error becomes smaller than 1% from 100 repetitions and higher (note that the actual threshold of 1% is somewhere between 50 and 100 repetitions). Based on these findings, we select 100[16] as the number of repetitions to be used for our validation step.

Table 3.8. Descriptive statistics for key performance measures obtained from the Markov Chain with different number of repetitions and fixed length of 50,000 state-trace records

	Server utilization		Average waiting time		Average queue length	
Repetitions	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)
10	0.88 (0.00)	0.00 (0.37)	42.91 (0.54)	1.06 (2.46)	4.31 (0.06)	0.11 (2.59)
20	0.88 (0.00)	0.00 (0.23)	43.38 (0.33)	0.66 (1.51)	4.34 (0.04)	0.08 (1.74)
50	0.88 (0.00)	0.00 (0.19)	42.71 (0.26)	0.51 (1.20)	4.28 (0.03)	0.06 (1.34)
100	0.88 (0.00)	0.00 (0.11)	43.25 (0.17)	0.33 (0.76)	4.33 (0.02)	0.04 (0.83)
200	0.88 (0.00)	0.00 (0.09)	43.44 (0.13)	0.25 (0.57)	4.35 (0.01)	0.03 (0.62)
500	0.88 (0.00)	0.00 (0.05)	43.64 (0.07)	0.14 (0.33)	4.37 (0.01)	0.02 (0.36)

Preliminary analyses: Selection of minimum state-trace length

Table 3.9 shows the mean performance measures and their margins of errors for the second set of preliminary analyses. This table shows that with longer traces containing a higher number of state-trace records, the margin of error becomes smaller and the level of precision of our estimate increases. For the KPI server utilization, a high level of precision (i.e., below 1%) is already achieved with a state-trace as short as 1,000 records.

[16] Note that 100 repetitions mean both the training and validation sets have 50 state-traces in each because we decided to split the complete set of state traces equally.

For the average waiting time and average queue length measures, the margin of error reaches <1% from 50,000 state-trace records and higher (note that the actual threshold of 1% is somewhere between 20,000 and 50,000 state-trace records). Based on these findings, we select 50,000 as the state-trace length to be used for our validation step, in conjunction with the 100 repetitions selected in the first set of preliminary analyses.

Table 3.9. Descriptive statistics for key performance measures obtained from the Markov Chain with varying state-trace lengths and fixed number of 100 repetitions

	Server utilization		Average waiting time		Average queue length	
Length	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)
1,000	0.88 (0.00)	0.01 (0.84)	43.06 (0.86)	1.68 (3.90)	4.31 (0.11)	0.22 (5.20)
2,000	0.88 (0.00)	0.00 (0.49)	42.90 (0.66)	1.29 (3.01)	4.31 (0.08)	0.15 (3.52)
5,000	0.88 (0.00)	0.00 (0.36)	43.51 (0.55)	1.07 (2.46)	4.28 (0.06)	0.12 (2.73)
10,000	0.88 (0.00)	0.00 (0.24)	43.34 (0.35)	0.69 (1.58)	4.36 (0.04)	0.07 (1.69)
20,000	0.88 (0.00)	0.00 (0.17)	43.16 (0.24)	0.48 (1.11)	4.33 (0.03)	0.05 (1.17)
50,000	0.88 (0.00)	0.00 (0.11)	43.25 (0.17)	0.33 (0.76)	4.33 (0.02)	0.04 (0.83)

Validation

Now that we have selected the number of repetitions and state-trace length that provides our desired level of precision, we can validate our results using inferential statistics. Table 3.10 presents results of the Student's *t*-test to test the null hypothesis H_0: $M_{M/M/1} = M_{MarkovChain}$ for each of the three performance measures. For all three performance measures, Student's *t*-test with $\alpha = 0.05$ was non-significant, indicating that there were no significant differences between the mean performance measures calculated from the Markov Chain and those calculated from the M/M/1 model.

Table 3.10. Student's *t*-test results for comparing mean performance measures obtained from M/M/1 model and Markov Chain for 100 repetitions and 50,000 state-trace length

	M/M/1 model		Markov Chain			
	M	*SE*	*M*	*SE*	$\lvert\Delta M\rvert$	*t*(df), *p*
Server utilization	0.88	0.00	0.88	0.00	0.00	1.27(198), $p = .207$
Average waiting time	43.64	0.30	43.25	0.17	0.39	1.14(155.32), $p = .256$
Average queue length	4.37	0.03	4.33	0.02	0.04	1.13(163.22), $p = .261$

To further validate our Markov Chain output, we compare the distributions of performance measure scores obtained from the Markov Chain and the M/M/1 validation set. Figure 3.6 shows the box and whisker charts for each KPI. We see that for the chosen number of repetitions (100) and state-trace length (50,000) and for each of the three KPIs, the box plots of the M/M/1 and Markov Chain overlap, indicating high similarity between the two distributions of scores. Likewise, both median lines lie within the overlap of the two boxes. It's noteworthy that the whiskers and boxes of the Markov Chain are consistently shorter than those of the M/M/1 validation set. This indicates that the data from the Markov Chain is less scattered and more consistently hovers around the median.

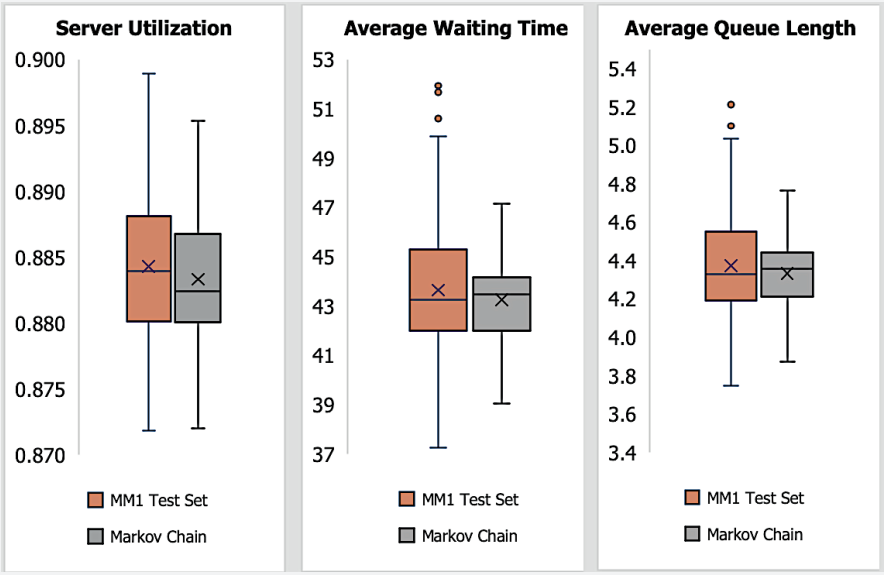

Figure 3.6. Box and whisker plots for 100 repetitions with the state-trace length of 50,000 state-trace records per repetition

Another test that we can perform to validate our Markov Chain results is the chi-square test to compare the frequency distributions of the state transitions obtained from the M/M/1 model and the Markov Chain. To do so, we use state-trace data containing the variables shown in Table 3.5 in Case Study Application §3.2.1.5. The histogram in Figure 3.7 shows the frequencies of state-transitions side-by-side for the M/M/1 model and the Markov Chain.

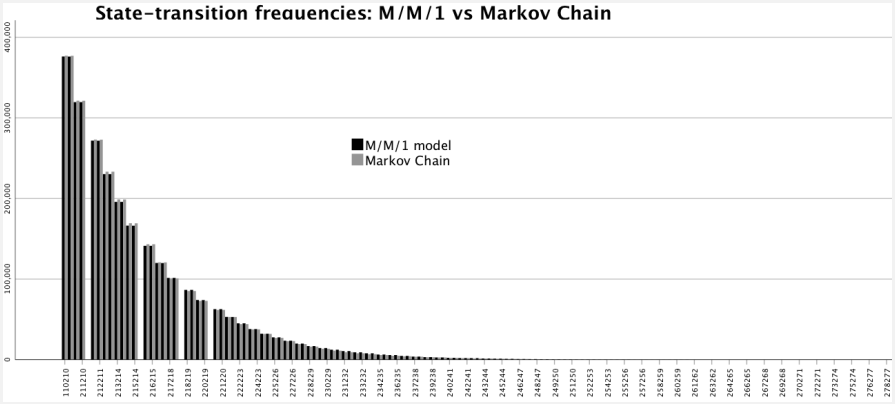

Figure 3.7. The frequency distributions of the state transitions of M/M/1 model and Markov Chain for 100 repetitions with 50,000 state-trace length

As can be seen from this histogram that the distribution has a very long right tail with low frequencies due to low probabilities of transitions happening for the higher queue lengths. In fact, the probability that the M/M/1 system has 21 entities or less (i.e., there are 20 or less entities in the queue and 1 entity is in the server) is

$$(1 - P_N) = (1 - a^N) / (1 - a^{N+1}) \text{ where } a = \boldsymbol{\lambda}/\mu$$

$$(1 - P_N) = (1 - 0.85^{21}) / (1 - 0.85^{22}) = 0.9671 / 0.97120 = 0.99492.$$

The probability that the M/M/1 system has 22 or more entities = $P_{N=}$ 1 - 0.99492 = 0.00508. Because their low probability (less than a percent), there is a much smaller chance that they occur during a simulation run. This could result in those rare state-transitions occurring in the Markov Chain-generated state-trace data (because of their presence in the training data), but not in the validation set (or the other way around). The consequence of this situation is that there may be a lot of cell frequencies in the two-way cross-tab that are small (or zero) relative to the expected counts in the chi-square test, which will inflate the chi-square value. We limit our queue length to 20 by excluding transitions beyond 230230 to overcome this shortcoming. The histogram in Figure 3.8 shows the frequency distributions after the recoding.

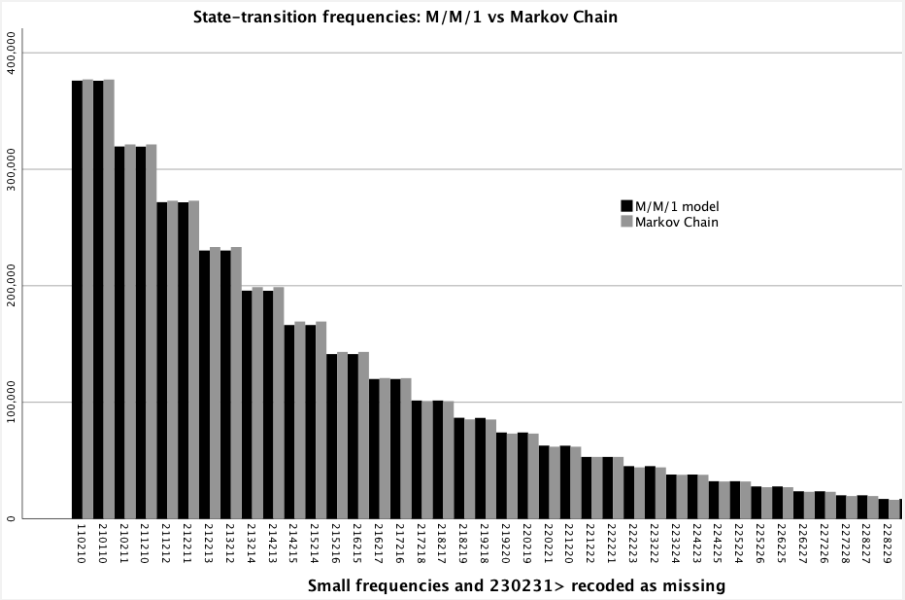

Figure 3.8. The frequency distributions of the recoded state transitions of M/M/1 model and Markov Chain for 100 repetitions with 50,000 state-trace length

Chi-square test results for the recoded dataset suggest that there is a significant difference between the two frequency distributions, $\chi^2(41, N = 9{,}681{,}935) = 375.11$, $p < .001$ with an effect size Cramer's $V = .006$. The significant test result is likely to be influenced by the large sample size (9,681,935); however, the effect size shows that the difference is negligible.

3.3 Conclusions

In this chapter, we presented a novel temporal data mining-based method for discrete-event simulation model abstraction using state-trace data. We started the chapter with the formal description of the key concepts of a state-trace and a state-trace record. We explained the EMMA (Episode Mining using Memory Anchor) algorithm and highlighted the reasons why the EMMA algorithm is one of the best suited frequent episode mining algorithms for our method. We provided a formal definition of Markov Chains and Markov Chains related concepts, such as the transition probability matrix and the

initial distribution vector. We explained the Markov property and emphasized why the memoryless property of Markov Chain states is akin to the discrete-event model states (Seo et al., 2018; Zeigler et al., 2018) and forms the backbone of our temporal data mining-based method to abstract discrete-event models.

We then presented the step-by-step approach of our temporal data mining-based method, in which we addressed several considerations and actions for the modeler. Within this detailed breakdown of our method, we created subcategories to further illustrate the factors that affect (a) the content of raw state-trace data, (b) the type of techniques and algorithms to use for preprocessing raw data for the frequent episode mining step, (c) the parameters and processes to generate and simulate the Markov Chain, and (d) the algorithms and techniques to validate the Markov Chain generated results against the original model generated validation set. Figure 3.9 summarizes all steps and considerations of our method.

To demonstrate and validate our proposed method, we presented an M/M/1 queueing system case study. The results of our case study, and specifically the validation process, show that we are able to obtain Markov Chain estimations of the three performance measures with a precision of <1% and that these estimates do not significantly differ from the original model's validation set. In addition, the probability distributions of the state-transitions obtained from the Markov Chain-generated state-trace data and from the validation set are highly similar. Although the χ^2-test – likely affected by the large sample size – indicated a significant difference between the two distributions, the Cramer's V effect size and graphical visualizations suggested that the magnitude of this difference is very small.

In sum, this chapter provides a first demonstration that our method can automate the process of model abstraction of discrete event simulation models by applying frequent episode mining techniques to state-trace data to generate a Markov Chain, and that this Markov Chain is capable of adequately estimating the stochastic behavior of the original model. In the next chapter, we will apply our method to a larger and more complex DEVS model.

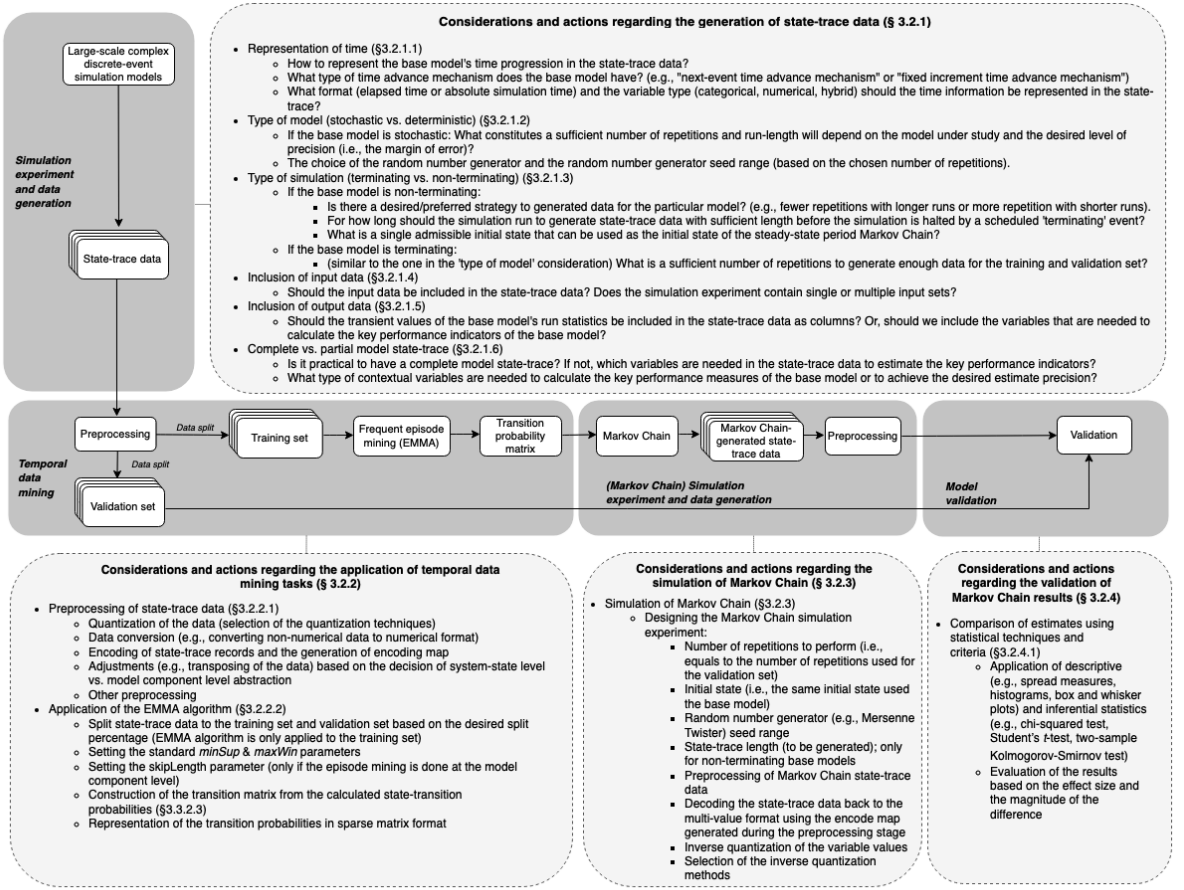

Figure 3.9. Automated DEVS model abstraction method and organization of the considerations and actions in Chapter 3

CHAPTER 4

Automated Discrete-event Model Abstraction: Application to Larger-Scale Models

This case study used in this chapter is largely based on:

Tekinay, C., Seck, M. D., & Verbraeck, A. (2012). Exploring multi-level model dynamics: performance and accuracy (WIP), *2012 Symposium on Theory of Modeling and Simulation Proceedings,* Article 20, 1-6. https://dl.acm.org/doi/10.5555/2346616.2346636

4 Automated Discrete-event Model Abstraction: Application to Larger Scale Models

In Chapter 3, we demonstrated the application of the automated model abstraction method on an M/M/1 queueing model. This chapter aims to illustrate how to apply the method introduced in §3.2 to a relatively larger, more complex model with different model characteristics, and extend the method where needed.

4.1 Battlefield Case Study

Operational challenges of the U.S. Military have been one of the core case studies of multi-resolution modeling (Davis & Bigelow, 1998; 2003; Petty et al., 2012). As Zeigler (2019) reconfirms, some of the primary reasons for using military case studies are the pioneering efforts of Davis and Bigelow, advocating the use of a family of models at multiple levels of abstractions when executing Department of Defense (DoD) simulation projects. A typical defense-related MRM scenario investigates the operational differences between the low-level (i.e., high-resolution) military combat units such as individual tanks and the aggregated high-level (i.e., low-resolution) combat units (e.g., battalions or tank platoons), when conducting tactical movements in a battlefield. The battlefield case study model, which is defined in Dynamic Structure Discrete-event System specification (DSDEVS; Zeigler et al., 2000, p. 235), explores a similar military scenario; a tank platoon maneuvers through a mountainous terrain to reach a tactical end station. An earlier version of the Battlefield DEVS[17] model was first introduced in (Tekinay et al., 2012). However, the model used in this dissertation contains several structural and behavioral changes to make the model better serve the further development of the automated model abstraction method.

In what follows, we provide the conceptual design of the battlefield model.

---

[17] Although the Battlefield DEVS model has a cell-based implementation, it is not implemented using the Cell-DEVS Formalism.

4.1.1 Scenario Description

The battlefield system modeled in this case study consists of a terrain with hills and passages between hills, and a tank platoon. A tank platoon is a military unit that consists of a team of 4 battle tanks, organized into two equal sections, A and B, with two tanks in each (Department of the Army, 2019). One of the main ideas behind forming a tank platoon is to eliminate, or at least reduce, the vulnerabilities of a single tank when facing enemy forces or moving through unfavorable terrain (Department of the Army, 2019).

The scenario simulated in this case study aims to assess the mission performance metrics of each platoon section, such as the average speed of a section or the total number of moves per section during a night mission with limited communication and visibility. The platoon's mission begins at the base camp located northwest of the terrain. The mission is completed when both sections reach an end-station in the southeast, only after successfully passing through three intermediate checkpoints within the hilly terrain (see Figure 4.1.a and 4.1.b). The terrain has an area of 10,000,000 m^2 and the lowest and the highest elevation is 144 meters and 375 meters, respectively.

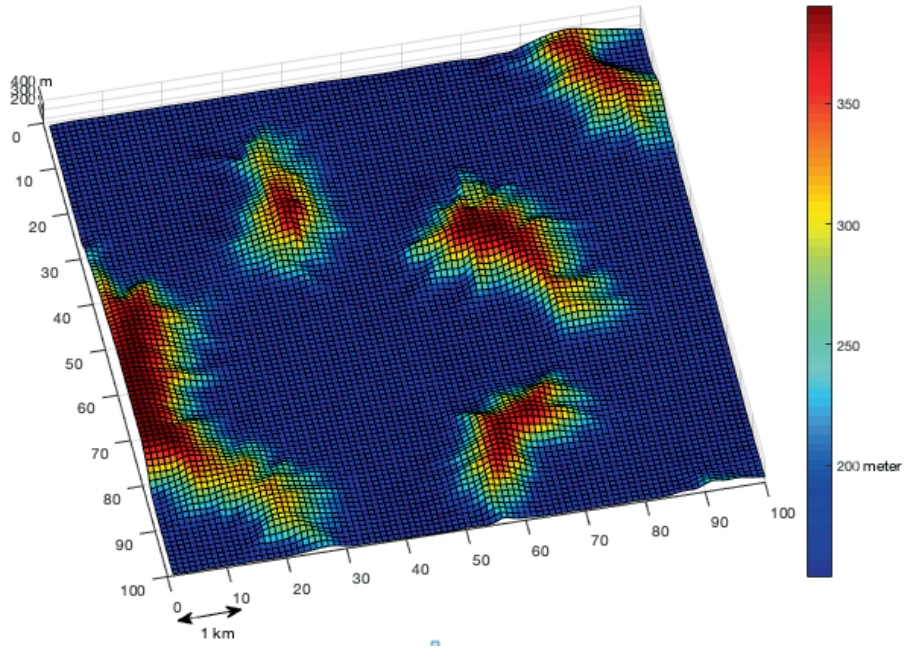

Figure 4.1.a. The hilly terrain used in the Battlefield case study

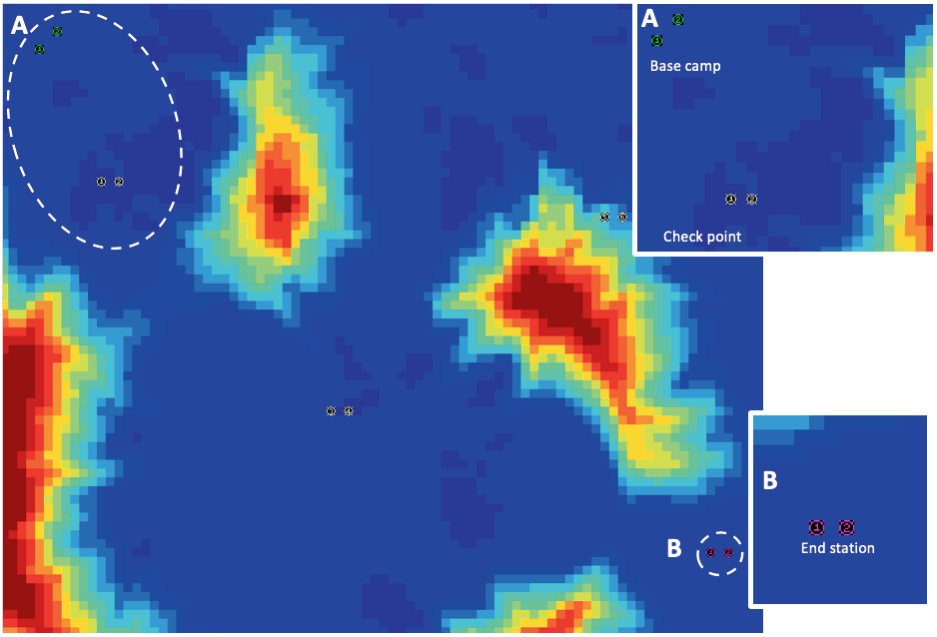

Figure 4.1.b. Base camp, check points and end station locations on the terrain

Both sections have equal roles and responsibilities[18], but the battle tanks in Section A and Section B have different technical specifications[19]. Section A consists of lighter tanks weighing 60 tons and a higher top-speed of 13.9 m/s, whereas the heavy armored Section B tanks weigh 64 tons and can get up to 11.8 m/s. The weight difference allows Section A to have quicker acceleration and both sections can climb slopes up to a maximum of 60%. The *percentage*, or *percent grade* of a slope can negatively or positively affect a section's speed. The percentage grade (%) of a slope is calculated by dividing the elevation change

[18] This is different from the real-life military setup (Department of the Army, 2019), where one section is the platoon leader and bears full responsibility for the mission's success and failure and the specifications of the tanks might be different.

[19] The specifications of the tanks are loosely based on the specifications of an M1A2 Abrams Main Battle Tank Retrieved February 11, 2022, from https://man.fas.org/dod-101/sys/land/m1.htm.

(i.e., rise) by the horizontal distance covered (i.e., run), and then multiplying the result by 100.

Sections are only given the coordinates of the checkpoints and the end station and there is no terrain reconnaissance information available to them. For the safety of the platoon and the success of the mission, sections are given a mission briefing to reach the end station as quickly as possible. However, due to the environmental and operational limitations, as well as equipment constraints, choosing the shortest route may not always be possible. These limitations and constraints are listed below[20]:

(I) *Formation compliance:* Sections should keep a minimum of 100 meters distance to each other to have a certain level of freedom and a maximum of 1,000 meters distance to provide some unity to maintain security and increase firepower.

(II) *Limited communication:* Sections can communicate with each other using radio. However, radio communication is limited to exchanging *instant coordinate updates* and *checkpoint arrival messages* between sections, and transferring additional information, such as reconnaissance data, is not permitted.

(III) *Reduced visibility:* While moving across the terrain, each section simultaneously conducts a *route reconnaissance*, or *recon*, to collect information about the surrounding landscape. This 360° reconnaissance is crucial to detect and avoid steeper slopes, which would otherwise cause the section to slow down. However, the reduced visibility caused by darkness, dense woods, or restricted terrain limits sections' capability to survey an area with a radius of more than 100 meters.

(IV) *Maximum allowed altitude:* Sections are not allowed to climb higher than 200 meters altitude in order to avoid detection.

(V) *Checkpoints:* Sections must pass through three intermediate checkpoints before arriving at the end station. The mission, however, requires both sections to have

[20] It should be noted that this is a stylized case and therefore different from a real-life military setup.

arrived at a checkpoint before proceeding to the next checkpoint. If a section arrives at a checkpoint and the other has not yet arrived, the first section waits for the other section's arrival. Only after the second section arrives and transmits a checkpoint arrival message over the radio will the first section resume movement.

Every 100 meters, the section that needs to update its route for the next 100 meters would send a coordinate request to the other section over the radio, to which it would receive an instant response. Then, in conjunction with the reconnaissance, determines the route for the next 100 meters that do not violate the maximum section distance within the formation and the maximum altitude requirements mentioned above. This way, sections can plan their routes in 100-meter increments without stopping.

In what follows, we will describe the conceptual model of this Battlefield system through a collection of assumptions and simplifications of the stylized system's operation.

4.1.2 Battlefield Conceptual Model

The 10 million m^2 hilly terrain is modeled using a heightmap with 100x100 meter patches or cells. A heightmap is an image where each pixel's color intensity corresponds to a surface elevation value. The lowest intensity pixel represents the lowest elevation, while the highest intensity pixel represents the highest elevation. An increase or a decrease of 1 in the pixel intensity equals an increase or a decrease of 10 meters in the elevation (i.e., *rise*). Each pixel of the heightmap represents a square cell in the 100x100 grid-based terrain with a surface area of 1 hectare (ha) or 10,000 m^2. Because of this pixel-to-area projection, we make the necessary assumption that the elevation value derived from a single pixel's intensity corresponds to the average height of the related square cell area.

Each cell on the grid is assigned a (*row, column*) coordinate. For instance, the top left cell has the coordinate (1,1) and the cell on its right is has the coordinate (1,2). The coordinate of the cell at the bottom right corner of the grid-terrain is (100,100). The distance between two cells (i.e., run) is the L_1 distance, or Manhattan distance, which is the sum of the absolute differences between the cartesian coordinates of the virtual center of each cell.

One of the simplifications we made when modeling the tank platoon was to represent platoon behavior at the section level rather than the individual tank level. As a result, the platoon is modeled as two sections, and sections are modeled as a single military unit instead of two individual battle tanks. Furthermore, recall that the tanks in Section A and Section B have different technical specifications. Sections in our model are designed to have the same specifications as the individual tanks of which they are composed. Another simplification we implemented when modeling the sections was to limit their ability to survey and, therefore, move only to four cardinal directions: north, east, south, and west. As a result, reconnaissance data is limited to the *neighbor cells* that are orthogonally adjacent to a sections' *host cell*[21], and it contains the average height of the four neighbor cells and their occupancy statuses (i.e., whether any one of the orthogonally adjacent cells is occupied by the other section at the moment of the reconnaissance). The decision to model the terrain as a square grid of 100m-by-100m cells together with the simplifications on the tank platoon's dynamic representation and reconnaissance behavior allows for a number of other simplifications:

(I) Sections moves only one cell at a time.

(II) Forbidding sections from sharing the same cell would automatically prevent them from violating the formation compliance's minimum distance rule.

When determining their route (i.e., the next cell to occupy), sections always prioritize the neighbor cells in the direction of the directional vector (depicted as green-gray cells in Figure 4.2.a and 4.2.b). We will use the term *preferred cells* for these prioritized neighbor cells for the remainder of this chapter.

[21] For the remainder of this chapter, the term "host cell" is used to refer a cell that is being (temporarily) occupied by a section.

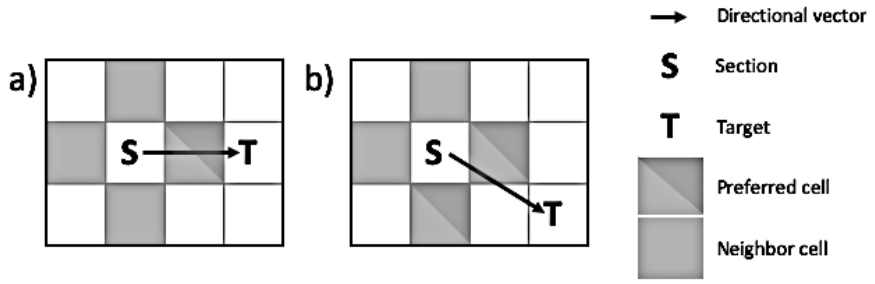

Figure 4.2. Preferred cells based on the directional vector of the section

To determine the directional vector of a section, the coordinate of host cell is subtracted from the target (i.e., a checkpoint or the end station). In the model, the directional vector is expressed as a cardinal or intercardinal direction for simplicity: north, east, south, west, north-east, south-east, north-west, and south-west. For example, the directional vector of the section in Figure 4.2.a is set as "east" and the directional vector of the section in Figure 4.2.b is set as "south-east". When a section's directional vector points intercardinal directions, the section considers checking the following four constraints before choosing one of the preferred cells to occupy (e.g., south or east cell in the case of the section in Figure 4.2.b):

(I) A preferred cell should not be occupied by another section.

(II) The movement of the section to a preferred cell should not cause a violation of the "formation compliance" constraint.

(III) The movement of the section to a preferred cell should not cause a violation of the "maximum allowed altitude" constraint.

(IV) The section should be able to climb up or down the percent grade of the slope between the host cell and the preferred cell(s).

Whether preferred or not, neighbor cells that do not comply with the four constraints listed above are removed from the reconnaissance data. When a section's directional vector points in a cardinal direction (e.g., "east" for the section in Figure 4.2.a) and the preferred cell in that direction is removed from the reconnaissance data, the section

prioritizes cells that are not located in the opposite direction of the directional vector. In Figure 4.2.a, for example, the section would initially prioritize the cells in the "south" and "north" over the cell in the west. On the other hand, when a section's directional vector points in an intercardinal direction (e.g., "south-east" for the section in Figure 4.2.b) and both preferred cells – in the direction of the directional vector's component vectors – are removed from the reconnaissance data, the remaining cells in the reconnaissance data would have the same priority. For instance, for the section Figure 4.2.b, the cells to the "west " and "north" will have the same priority if the preferred cells in the "east" and "south" of the section were removed.

If none of the preferred cells are eliminated and there are two preferred cells left to consider as shown in Figure 4.2.b, then the section should choose the cell that allows it to travel the fastest route. The check for the quickest route is done based on the percent grade of the slope between the host cell and each preferred cell. Sections begin their mission idle (the speed is 0 m/s) at their corresponding base-camp coordinates [(5,5), (5,7)]. The initial moving speed of each section is drawn from a Triangular distribution. For Section A, the triangular distribution is defined with a lower limit a = 5 m/s, upper limit b = 10 m/s and mode c = 7.5 m/s. For Section B, it is a = 3 m/s, b = 7 m/s and c = 5 m/s. To this value, we add noise drawn from a continuous uniform distribution within a range of -1 and 1. A section's moving speed is affected by the percent grade of the slope and each section's quantized speed values for various percent grades (see Figure 4.3) are given as an input to the model by design. For each recalculation of the section speed, we add Gaussian noise with a variance σ_s^2 of 1 to these quantized section speed values to simulate the effects of imperfections in the terrain's surface and various obstacles such as trees and rocks, as well as variations caused by the human (driver) factor on the sections' speed. Sections have different speeds during a mission (different initial moving speeds and changes in their speed based on the steepness of the terrain). They are designed to move from one cell to the next instantaneously at the time of their previously scheduled movement.

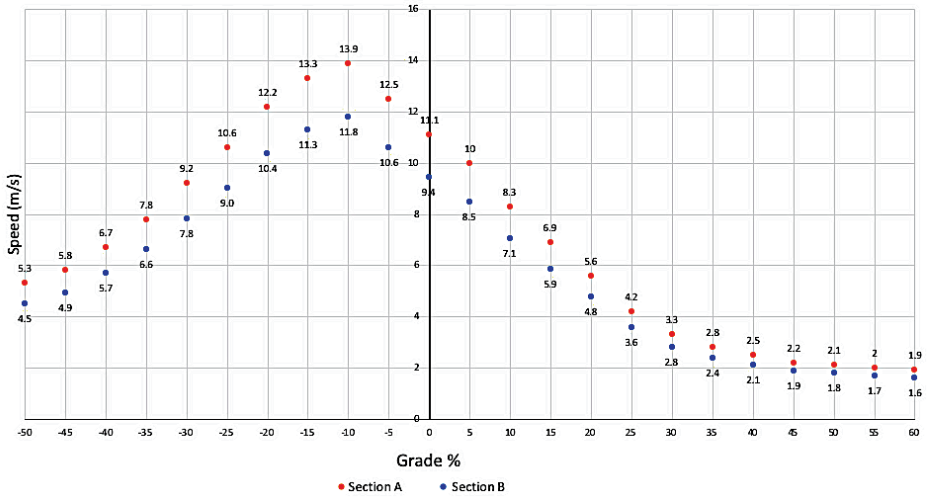

Figure 4.3. Quantized section speeds for various percentage grades

If the percentage grades of the slopes of the two preferred cells are also the same, then the sections employ a *tie-breaker mechanism*. The tie-breaker mechanism aids in the selection process by generating a random number u from a uniform distribution $\mathcal{U}_{[0,1]}$. The section makes its final decision based on the value (i.e., $u \leq 0.5$ or $u > 0.5$) of this number.

This sequential rule-based approach to select the next cell may result in a deadlock situation for a section, in which the section continues to move back and forth between the same cells. Such a deadlock could be caused by a temporarily eliminated cell as a result of being occupied by the other section at the time of the reconnaissance, or by the formation compliance constraint, that is the maximum allowed L_2 distance (or Euclidean distance) between each section. In these cases, the deadlock may be broken after the section repeats the same moves several times. However, such a deadlock may be caused by a permanently eliminated cell (or cells) as a result of the maximum altitude constraint or an extreme percentage grade of the slope. As a result, the impasse is permanent.

To deal with temporary deadlocks and avoid permanent ones, each section is allowed a user-specified maximum number of the same moves. This number may differ for each section. For instance, because the faster section is more likely to violate formation

compliance constraint and may have higher number of repetitive moves. If the deadlock persists after several repetitions, the neighbor cell that is causing the deadlock situation is removed from the reconnaissance data to temporarily eliminate the repetition. Once the problematic neighbor cell is removed from the reconnaissance data, the section would (as per usual) choose the next best cell from the list of available neighbor cells. This is referred to as a *forced move*. In rare cases, a section may not have any available neighbor cells to select, after temporarily removing the problematic cell from the reconnaissance data. Only in this case is it permissible for a section to disregard the maximum altitude constraint and move to a cell with an average height (altitude) value greater than the maximum altitude limit. When the section moves to a cell with a height less than the user-defined maximum altitude limit, the constraint is re-imposed on it.

4.1.3 Modeling the Battlefield case in the DEVS Formalism

The conceptual model described in §4.1.2 is implemented in accordance with the guidelines provided by Seck and Verbraeck (2009) for implementing hierarchical DEVS models in DSOL. The model consists of two atomic components; cell and section as shown in Figure 4.4.a and Figure 4.4.b, and two coupled components; terrain and platoon as shown in 4.5.a and 4.5.b respectively.

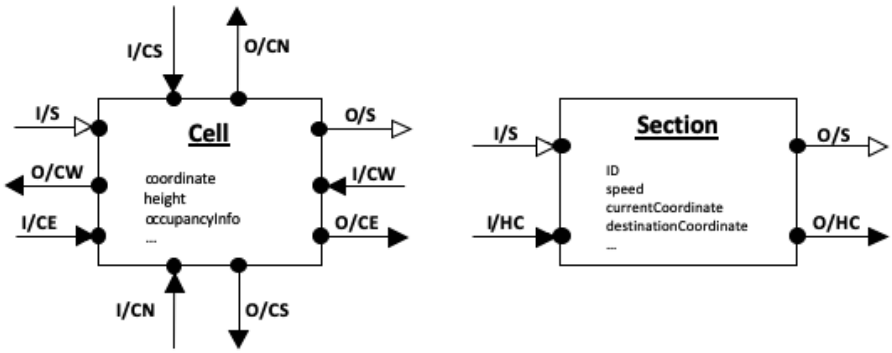

Figure 4.4. Cell and Section atomic components

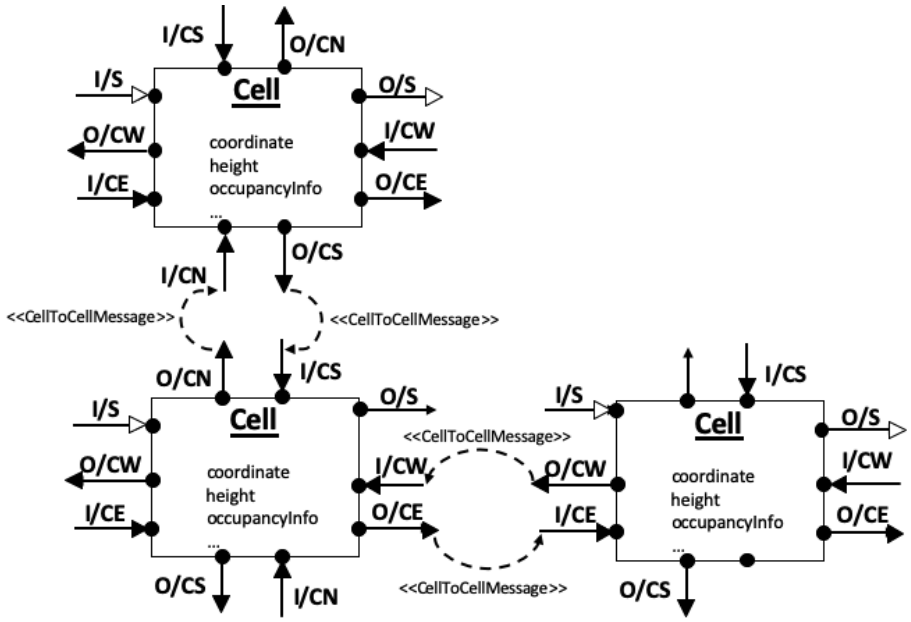

Figure 4.5.a. Terrain coupled model

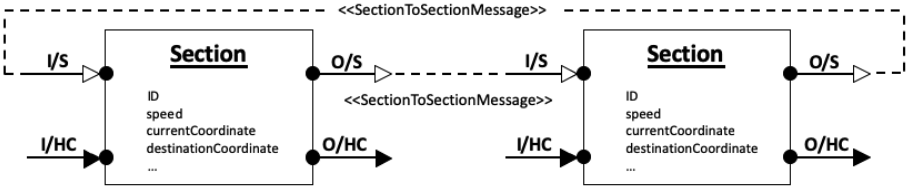

Figure 4.5.b. Platoon coupled model

The terrain coupled model shown in Figure 4.5.a is composed of cell-based atomic model components[22]. Cell-to-cell couplings are statically added at the beginning of the simulation between the corresponding I/O ports (see Table 4.1) of cell components, forming a grid-based network of connected cell atomic model components. Cell components located at the corners of the two-dimensional terrain have only two cell-to-cell couplings because they only have two non-diagonal neighbor cells, while the cells at the edges of the terrain have three cell-to-cell couplings, and all the other cells in the terrain have four cell-to-cell couplings.

Table 4.1. I/O ports of the cell atomic DEVS models in the Battlefield model

Cell atomic model input ports		Cell atomic model output ports	
I/CE	Input/Cell East	O/CE	Output/Cell East
I/CW	Input/Cell West	O/CW	Output/Cell West
I/CN	Input/Cell North	O/CN	Output/Cell North
I/CS	Input/Cell South	O/CS	Output/Cell South
I/S	Input/Occupying Section	O/S	Output/Occupying Section

Similarly, the platoon coupled component shown in Figure 4.5.b is modeled by statically adding the section-to-section couplings between the corresponding I/O ports (see Table 4.2) of the two sections.

Table 4.2. I/O ports of the section atomic DEVS model in the Battlefield model

Section atomic model input ports		Section atomic model output ports	
I/S	Input/Section	O/S	Output/Section
I/HC	Input/Host Cell	O/HC	Output/Host Cell

[22] We use the term "model component" referring to the atomic and coupled model in the hierarchical DEVS formalism (Zeigler et al., 2000).

Unlike the static cell-to-cell and section-to-section couplings, section-to-cell couplings are established dynamically (see Figure 4.6) according to the Dynamic Structure DEVS specifications (Zeigler et al., 2000). The only exception to this is the initial couplings of the sections to their corresponding pre-determined base-station cells, which are added at a certain instance in beginning of the simulation. After the initial coupling is established between a section and its base-station cell, and the section is ready to move (once it completes its preparations), the internal coupling with the host cell is removed using the *removeInternalCoupling* method (lines 1155-1157 in Figure 4.7) and a new internal coupling is added to the new (host) cell using the *addInternalCoupling* method (lines 1159-1161 in Figure 4.7). This is repeated until the section reaches its destination.

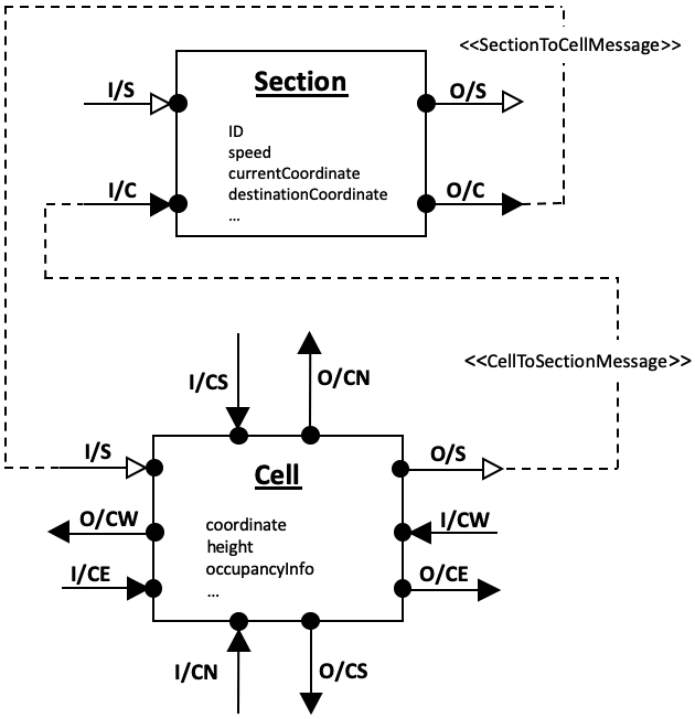

Figure 4.6. Dynamic coupling of a section to a cell via its I/O ports

```
1155    //section is removing the internal coupling from the previous host cell
1156    this.parentModel.removeInternalCoupling(this.ToCell, currentCellInputPort);
1157    this.parentModel.removeInternalCoupling(currentCellOutputPort, this.FromCell);
1158
1159    //section is adding the internal coupling with the new host cell
1160    this.parentModel.addInternalCoupling(this.ToCell, nextCellInputPort);
1161    this.parentModel.addInternalCoupling(nextCellOutputPort, this.FromCell);
1162
1163    //section is introducing itself to the new host cell
1164    //So that, the new host cell can update its occupancy info and update its neighbors
1165    this.ToCell.send(new SectionToCellMessage(this.sectionID.false. true, false, false, false));
```

Figure 4.7. Dynamic coupling between a section and old and new host cells

For each atomic component the behavior is structured into several different phases (Honig and Seck, 2012; Zeigler, 2000, p. 214). Each phase is identified by a name that indicates the type of the activity of the component, and a lifetime that describes the duration the entity remains in that particular phase. It should be noted that the phase variables are ordinary state variables *S* of the atomic model according to the specification in §2.2.4.1. Below, Table 4.3 lists these atomic components, their corresponding phases, lifetime of each phase, and a concise description of each phase, which designates the model component's activity.

Table 4.3. Phases of the atomic components in the Battlefield DEVS model

Type	Phase	Lifetime	Description
Cell	EMPTY	$+\infty$	The cell is not occupied by a section
	OCCUPIED	$+\infty$	The cell is occupied by a section
	UPDATE	0 (zero)	Updating the orthogonally adjacent cells with the status info
Section	INITIALIZATION	t_{init}	Initialization of the sections
	OCCUPY	t_{occupy}	Occupying the (chosen) next host cell and establishing the new couplings
	START_MOVING_AGAIN	t_{occupy}	Moving back after both sections arrived at a checkpoint
	INVESTIGATE_CELL_STATUS	0 (zero)	Investigating the status (e.g., occupancyInfo, height) of orthogonally adjacent cells

REQUEST_SECTION_COORDINATES	0 (zero)	Requesting the coordinates of the other section
SEND_FORMATION_TO_SECTIONS	0 (zero)	Sending the requested coordinates to the other sections
WAIT_SECTION_COORDINATES	+ ∞	Waiting for the coordinates of the other section
SELECT_CELL_TO_OCCUPY	0 (zero)	Updating the current and the next host cells (for dynamic coupling)
DESTINATION_REACHED	+ ∞	Arriving at the end-station

Each atomic model component (cell and section) has phases with zero lifetime, for example, Cell.UPDATE and Section. INVESTIGATE_CELL_STATUS. These phases are used to request information from other atomic components or to update their information stored at other atomic components via the corresponding I/O ports. For example, in our model, a section conducts a reconnaissance and collect information about the neighbor cells via its host cells. To do that, the section will first transition from START_MOVING_AGAIN to INVESTIGATE_CELL_STATUS. Because the INVESTIGATE_CELL_STATUS has a zero life-time, a message of type <SectionToCellMessage> is sent to the host cell via the corresponding output port O/HC (Table 4.2) of the sections, to be received by the corresponding input port I/S (Table 4.1) of its host cell.

4.2 Application of the Temporal Data Mining-based Model Abstraction Method to the Battlefield DEVS Model

In the following subsections, we illustrate step-by-step how we apply the method introduced in §3.2 to the Battlefield DEVS model in the order presented in Chapter 3 (see Figure 3.9).

4.2.1 Generation of DEVS Model State-Traces

Recall from §3.2.1 that we divided the considerations when generating state-traces from a simulation model into six categories: *representation of time* (see §3.2.1.1), *the type of model*

(stochastic vs. deterministic) (see §3.2.1.2), *the type of simulation (terminating vs. non-terminating)* (see §3.2.1.3), *the inclusion of input variables* (see §3.2.1.4), *the inclusion of output variables* (see §3.3.1.5), and *the complete model state-trace vs. partial model state-trace* (see §3.2.1.6).

4.2.1.1 Representation of Time

When generating the state-trace data for the Battlefield case study, the DEVS simulator logs state-trace records at each event occurrence rather than recording at fixed time intervals. The "fixed-increment" time advance mechanism is not suitable for the Battlefield case study for two reasons: (a) We would like to include state-variables for multiple model components in the state-trace data, with the goal of capturing every state transition that these selected model components go through. These state transitions of the model components are triggered by the internal and external events. In the Battlefield DEVS model, the lifetimes of states (see Table 4.3) have significant variations. (b) Sections' movements are not synchronized. Section types A and B have different speed characteristics as well as different initial moving speeds, and their speed is affected by the terrain (e.g., the steepness of slopes) and by the added noise. As a result, using fixed time-steps to generate state-traces may result in an inaccurate representation of the model behavior. Some transient states, for example, may be underrepresented in the data due to event skipping at larger step sizes. At smaller step sizes, however, some of the states such as formation requests or status updates, maybe be overrepresented. Therefore, an event-based representation of time will provide a better representation of all states and all state changes with the correct frequencies of occurrence.

Another important consideration regarding the inclusion of time is the decision of the format of time (see §3.2.1.1). The time recorded in the state-trace data is the elapsed time. The main reason behind choosing elapsed time over the monotonously increasing absolute simulation time is to ensure that the ability of frequent episode mining algorithms to find recurring patterns in data is not hindered by the continuously increasing absolute time.

4.2.1.2 Stochastic vs. Deterministic DEVS Models

The Battlefield DEVS model is a stochastic model with multiple random variables. These random variables have their own probability distributions. For instance, the initial moving speed of Section A is drawn from a standard Triangular distribution, with a lower limit a = 5 m/s, upper limit b = 10 m/s and mode c = 7.5 m/s.

To accurately estimate the stochastic behavior of the model and to collect a sufficient amount of data, we ran the same scenario 100 times with a different RNG seed value per run (see Table 4.4). Our aim was to equally split the data into two non-overlapping sets: a training and a validation data set. After the split, the total number of state-trace records in the training set is 32,610, whereas the total number of state-trace records in the validation set is 32,714.

Table 4.4. Random Number Generator seed value ranges for each dataset in the case study

Model	Data set name	# of state traces per dataset	Seed value range	Total state trace records (50 state traces)	Average state trace length
Battlefield (base model)	Training	50	1-50	32,610	652.20
	Validation	50	10,001-10,050	32,714	654.28

4.2.1.3 Terminating vs. Non-terminating Simulations

The Battlefield simulation is terminating. Each run terminates at a different run time (due to the model's stochasticity) when both sections arrive at their predefined end-station coordinates $S_A(68,82)$, $S_B(68,84)$. Thus, the simulation runs and the state-traces generated from these runs differ in length. In the validation stage in §4.2.4, we will investigate whether the Markov Chains runs have sufficiently close run-lengths compared to the base model's using descriptive statistics.

4.2.1.4 Inclusion of Input Data

The Battlefield DEVS model used in this case study has a single set of input data. Input variables of the case study model and their values are given in Table 4.5 below:

Table 4.5. Some of the Input variables of the Battlefield model and their values

Input variable	Value
Total number of sections	2
100x100 heightmap of the terrain	Figure 4.1.a; 4.1.b
Section A – Initial coordinate	(5,5)
Section B – Initial coordinate	(5,7)
Section A – End-station coordinate	(68,82)
Section B – End-station coordinate	(68,84)
Total number of checkpoints	3
Section A - Checkpoint 1 coordinate	(26,13)
Section B – Checkpoint 1 coordinate	(26,15)
Section A – Checkpoint 2 coordinate	(52,39)
Section B – Checkpoint 2 coordinate	(52,41)
Section A – Checkpoint 3 coordinate	(30,70)
Section B – Checkpoint 3 coordinate	(30,72)
Elevation change per 1-pixel intensity	10 m
Section A & B – Discrete speed profiles	Figure 4.3
Mersenne Twister seed range (for training data)	1 - 50
Mersenne Twister seed range (for validation data)	10,001 – 10,050
Total number of cells in a terrain	10,000
Single cell area	100 m^2
Total terrain area	10,000 m^2

The model has a single (fixed) set of input values and there is no external input. Therefore, the inclusion of input variable values in the state-trace data (as columns) does not affect the variability of the episodes discovered by the episode mining algorithm because they remain constant. Therefore, the input data is excluded from the state-trace data.

4.2.1.5 Inclusion of Output Data

Based on the input values presented in the previous section, the theoretical calculations of several performance indicators of the Battlefield model are given below in Table 4.6:

Table 4.6. Performance indicators of the battlefield model

Performance Indicator	Description	Example Values
Active mission time Section A	Total time it takes for Section A to reach its pre-determined end-station coordinate from the base station. The duration of the mission calculation excludes the waiting times at checkpoints	1,814.76 s
Active mission time Section B	Total time it takes for Section B to reach its pre-determined end-station coordinate from the base station. The duration of the mission calculation excludes the waiting times at checkpoints.	2,270.72 s
Total elevation climbed Section A	The sum of the positive elevation values that a section climbs during a mission.	626 m
Total elevation climbed Section B	The sum of the positive elevation values that a section climbs during a mission.	252 m

To estimate the performance indicators listed in above Table 4.6 using our method, the state and contextual variables to be included in the state-trace data are given in Table 4.7. Elapsed time is also part of the state-trace data used in the experiments (see §4.2.1.6 for more details on the experiments). The *active mission time* of a section is the sum of all elapsed times in the state-trace data, belonging to the movement of that particular section. However, the elapsed times recorded in the state-trace are the elapsed times for all events, that is moving and non-moving events (e.g., formation check and reconnaissance). As a result, we need contextual variables which would allow us to distinguish between Section A and B's moving and non-moving events. To solve that, the "movement indicator" contextual variable is captured in the data to flag events that are related to the sections' movements (e.g., "1" = moving). When a section is idle at the time of an event arrival, then the movement indicator variable value in the state trace-record is reported as "0"; i.e., a non-moving event. Therefore, to calculate the active mission time

of a section from the state-trace data, we sum elapse times of the "moving events" of the particular section.

Table 4.7. The state and contextual variables included in the state-trace

VariableName_SectionId	Description	Example Values
speed_A	Speed of section A in m/s	8.27
elevation_A	Elevation of the host cells of Section A in m	172
speed_B	Speed of section B in m/s	7.13
elevation_B	Elevation of the host cells of Section B in m	168
movementIndicator_A	A contextual variable for Section A and B to identify	1
movementIndicator_B	the moving events (1 = moving event, 0 = non-moving event)	0
distanceToCheckPoints_A	Euclidean distance to the next checkpoint (including the end-station) in m.	1,581
distanceToCheckPoints_B		1,708
checkpointID_A	Sequence ID of a check point starting from the	0
checkpointID_B	value 0 (e.g., 0, 1, 2). Check point ID > 2 indicates that the next station is the end-station.	0

Similarly, to calculate the *total elevation climbed* of a section from the state-trace data, we sum all positive differences between the two subsequent elevation values (i.e., $E_{n+1} - E_n$, where n is the sequence - or row - number) in the trace data.

4.2.1.6 Complete vs. Partial Model State-Trace Data

The battlefield simulation generates partial model state-trace data. A complete set of trace data, which contains the state variables for all model components and possible additional contextual variables would be impractical for the mining task due to its dimensions. For instance, the terrain consists of a total of 10,000 cell atomic components. Reporting every state variable for all 10,000 cells in the state-trace data at every event occurrence will have a negative impact on the resulting data size, which is the cartesian product of the number of cells and the number of state variables such as the occupancy statuses, elevations per cell component. Instead, a more reasonable way to report the model's state without

compromising the validity of the abstract model would be the partial reporting of the variables that are essential to capture the dynamic behavior of the model. Recall from §4.1.3 that only the host cells are part of the dynamic coupling and the host cells centralize the communication between the neighbor cells during the movement of sections. Furthermore, for the calculation of the performance indicator *total elevation climbed (for each section)* chosen for this case study, only the elevation data for host cells is essential to be reported as a part of the state-trace data.

Although a partial model state-trace containing a subset of the variables listed in Table 4.7 may suffice to generate a Markov Chain model with a sufficiently accurate estimation of base model's KPIs, we cannot be certain about it without testing this assumption. Therefore, we designed four experiments to assess the effects of (1) different number of variables included in the state-trace data (experiments 1 and 2), and (2) different levels of quantization of these fixed variables (experiments 3 and 4). For the first and second experiments, we created two datasets, each containing the same amount of state-traces (100) with an equal split of training (50) and validation (50) data (see Table 4.4). The difference between the two datasets is the content of the state-traces in terms of the variables included. In the first data set, we included speed, elevation and movement indicator variables for each section along with the elapsed time (used for MC1, see Table 4.8). The second dataset (for MC2) includes all the aforementioned variables and additionally the variable types "distance to check points" and "check point id" for each section (i.e., an additional 4 new columns). Although the inclusion of additional variables in the state-trace data may result in an increase in the number of unique episodes identified by the EMMA algorithm, a more precise account of the context provided by the additional variables for the state changes may improve the accuracy of Markov Chain. Because the battlefield is a terminating simulation, we hypothesize that the inclusion of variable types "distance to check points" and "check point id" may be necessary for the Markov Chain to accurately identify this terminating behavior. To observe the effects of different level of quantization of a fixed set of variables in experiments 3 and 4, we created two subsets of the MC2 data set with a lower and a higher level of binning. We hypothesize that the increased level of quantization may be necessary to further improve the accuracy of the MC2.

4.2.2 Application of the Temporal Data Mining Tasks to the State-Trace Data

Below, we discuss the battlefield case-study specific data challenges and our decisions regarding the selection and the application of the preprocessing methods.

4.2.2.1 Preprocessing of the State-Trace Data

After the completion of all runs, the next step in our method is the preparation of the data for the episode mining algorithm. Figure 4.8 shows the frequency distributions of the speed, elevation and distance to check point data for the training (Figure 4.8.a for speed, Figure 4.8.c for elevation, and Figure 4.8.e for distance to check point) and the validation sets (Figure 4.8.b for speed, Figure 4.8.d for elevation, and Figure 4.8.f for distance to check point).

As previously mentioned in §4.2.1.6, for experiments 3 and 4, we created two subsets of the MC2 data set with a lower and a higher level of binning. Among the variables in these subsets, elapsed time, speed and distance to check point are continuous variables as they take values from the set of non-negative real numbers $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$. Meanwhile, the elevation data of the host cells is defined as a discrete variable in the model and it takes values from the set of non-negative natural numbers $\mathbb{N}_{\geq 0} = \{x \in \mathbb{N} \mid x \geq 0\}$. All four variables types are quantized using an equal-width binning based method. For the lower resolution model, we chose bin widths 2, 20, 5, and 5 for the speed, elevation, elapsed time, and distance to check point variable types, respectively. Furthermore, for the speed and distance to check point variables, behaviorally significant values of waiting speed at checkpoints/end-stations ($\mathcal{V}_s$ = 0 m/s) and distances to checkpoints/end-stations (d_s = 0 m) are captured as separate bins. For the higher resolution model, we halved the bin width sizes. The values of the movement indicators and the check point ids are represented in the data without being pre-processed. The movement indicator variable takes value 0 or 1 to indicate the absence or presence of a movement for a section. The check point indicator takes values from the range [0-2] and therefore do not need to be quantized.

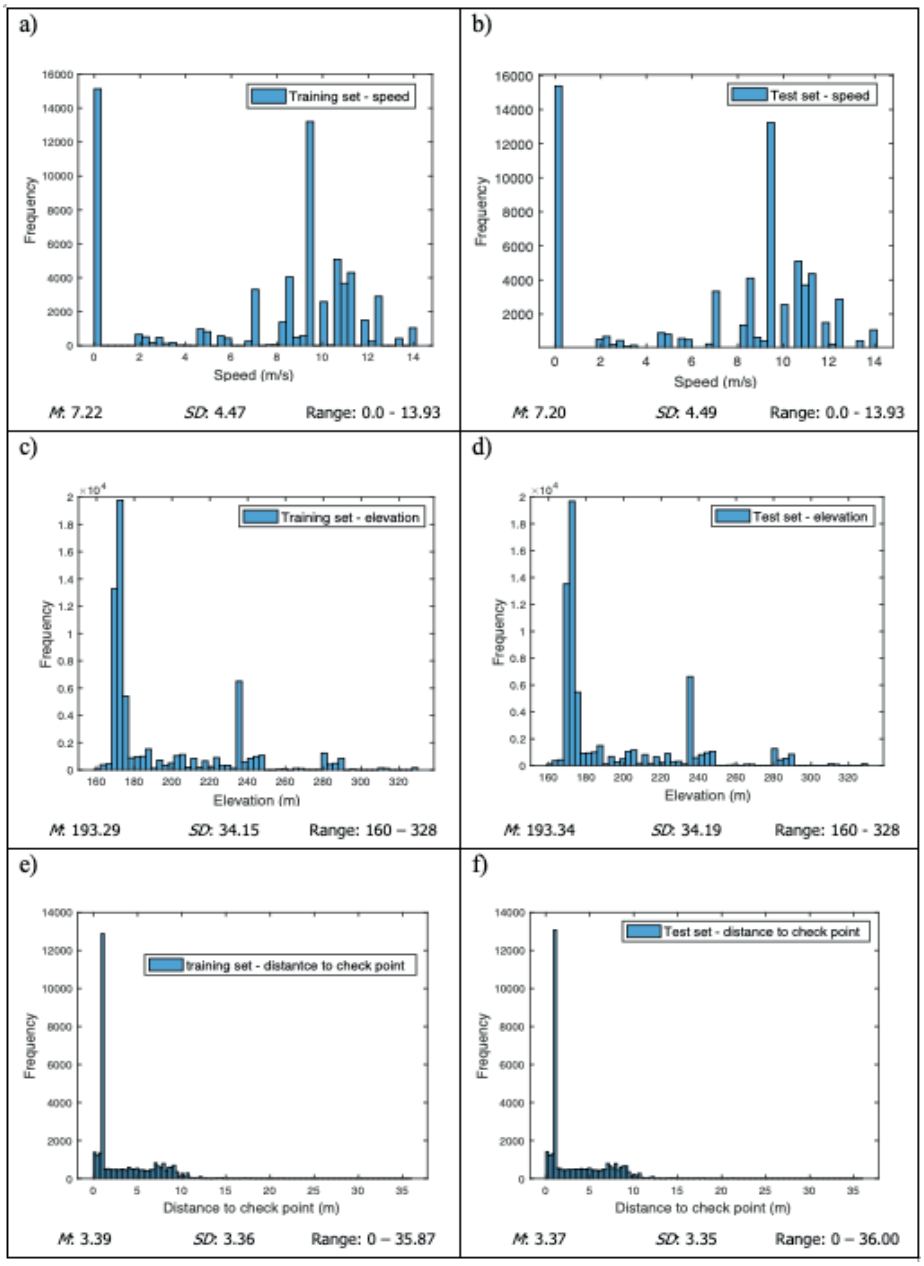

Figure 4.8. Frequency distributions of the speed, elevation and distance to check point variables for both the training and the validation sets

The final step of the data preprocessing for this case study is to encode the rows of the state-trace data. A single row of the tabular state-trace data corresponds to a state-trace record, which is a combination of variable values from multiple atomic models; where each variable is represented in a fixed column. Although the particular implementation of the EMMA algorithm used for this case study can process multiple positive integer values in a single row (each value separated by a single space), it assumes that these multiple values are sorted according to a total ordering (e.g., ascending) and with no repetition of the same value in the same row. This is because the SPMF implementation (Fournier-Viger et al., 2014; 2016) is intended to find frequent items (events) from a given itemset (event set). However, these assumptions are not valid for the state-trace records. A simple solution for this is to create a hash table that associates every unique state-trace record (multi-dimensional value) to an incrementing positive integer (key). This process generates two types of output: (a) State-traces that are compatible with the EMMA algorithm input format, and (b) a hash table which will later be used to decode the values in the Markov Chain generated state-traces back to the original (i.e., multi-value) format.

As a result, we will compare the four Markov Chains (MC1 – low resolution binning; MC1 – high resolution binning; MC2 – low resolution binning; MC2 – high resolution binning) based on the precision and the accuracy of their estimates for the base model's KPIs.

4.2.2.2 Application of the Frequent Episode Mining Algorithm to Preprocessed State-Trace Data

Following the preprocessing of variables in the training and validation data sets (for all 4 experiments), the next step in our method is the application of the EMMA algorithm to each experiment's training data set. The *maxwin* parameter of the EMMA algorithm is set to "2" so that the algorithm will identify (a) episodes with both window size "1", which corresponds to all unique states, or state-trace records, and (b) episodes with window size "2", which are the state-transitions, or two consecutive state-trace records. As explained earlier in §3.1.2, a higher *maxwin* value is not needed as the next state in DEVS Formalism is only dependent on the current state, the external inputs via the Ports (in this particular model), and nothing else. Furthermore, the *minsup* parameter is set to "1"

to include *all* frequencies of each unique state and state-transition across all 50 traces in the all four training sets. Using the obtained frequencies, the transitioning probabilities between all states are calculated and represented in the transition probability matrix that will generate the Markov Chain. It is important to mention that the inclusion of states with a frequency of 1 (*minsup* = 1) ensures the inclusion of essential but infrequent behavioral states (e.g., reaching the end station or the check point) in the Markov Chain's state set.

After the application of the EMMA algorithm to the training set from dataset 1 and 2, the number of unique states identified for four Markov Chains are: 915 states for MC1 – low resolution binning; 1,590 states for MC – low resolution binning; 1,708 states for MC1 – high resolution binning; 3151 states for MC2 – high resolution binning. Recall from Table 4.4 that the total number of state-trace records across all state-traces (50) in both dataset's training sets is 32,610.

4.2.3 Simulation of the Discrete-time Markov Chain

After the calculation of the state-transition probabilities and the generation of the transition probability matrices for all four Markov Chains, the next step of our method is to perform 50 runs with each Markov Chain and generate state-trace data from all four chains for the validation process. Unlike the M/M/1 case study in Chapter 3, the battlefield is a terminating simulation. To stop the generation of the state-trace data from the Markov Chain when it reaches the final state, we implemented a mechanism to terminate the simulation immediately after the arrival to the final state.

After all four Markov Chains completed their runs, we decoded the state-traces obtained from the simulation of each model. For the decoding, we used the hash table generated from the preprocessing of the data sets before the application of the EMMA algorithm. We then dequantized the values of the four variable types (i.e., speed, elevation, distance to checkpoints, and elapsed time) before the validation process by assigning the center values of their corresponding quantization bins.

4.2.4 Validation of the Markov Chain Results

Before comparing mean performance indicators obtained from the Markov Chain with those from the battlefield base model, we must first determine which of the four Markov Chains performs best in terms of precision. This can be assessed by calculating the margin of error of the estimates for each Markov Chain.

Table 4.8 presents the means *M* and the margin of error *MOE* generated from 50 runs of the base model and the Markov Chain models MC1 and MC2 obtained with low- and high-resolution binning. These data show that the margin of error for all KPIs is considerably lower in the MC2 models with the additional checkpoint indicators as compared to the MC1 models or the lower binning resolution. Specifically, in the lower resolution model, the margin of error reduces with 26 to 31 percentage points to 2.45%-5.04% across KPIs when the Markov Chain is trained with the additional checkpoint indicators. In the higher resolution model, the margin of error is even further reduced to 1.46%-4.59% across KPIs.

Table 4.8. Descriptive statistics for the performance indicators obtained from the Markov Chains for the four experiments

	Active mission time				Total elevation climbed			
	Section 1		Section 2		Section 1		Section 2	
	M (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)	*M* (*SE*)	MOE (%)
Original quantization levels (low resolution binning)								
Base model	1,060.30	6.55	952.70	6.66	417.20	14.47	368.00	18.16
	(3.34)	(0.62)	(3.40)	(0.70)	(7.38)	(3.47)	(9.27)	(4.94)
MC1	1,052.70	303.02	925.60	266.35	428.00	151.59	385.20	138.92
	(154.60)	(28.79)	(135.90)	(28.78)	(77.34)	(35.42)	(70.88)	(36.06)
MC2	1,064.85	28.56	964.55	23.66	421.60	16.76	369.60	18.62
	(14.57)	(2.68)	(12.07)	(2.45)	(8.55)	(3.97)	(9.50)	(5.04)
High resolution binning								
Base model	1,054.35	6.13	906.65	7.62	406.40	15.71	383.80	20.21
	(3.13)	(0.58)	(3.89)	(0.84)	(8.01)	(3.87)	(10.31)	(5.26)
MC1	1,056.50	370.86	918.38	319.33	422.00	165.83	392.20	156.30
	(189.21)	(35.10)	(162.93)	(34.77)	(84.61)	(39.30)	(79.74)	(39.85)
MC2	1060.30	16.45	912.98	13.29	402.40	15.87	397.20	18.24
	(8.39)	(1.55)	(6.78)	(1.46)	(8.10)	(3.94)	(9.31)	(4.59)

Note. MC1 = Markov Chain with duration, speed, elevation, movement indicator; MC2 = Markov Chain with duration, speed, elevation, movement indicator, checkpoint indicators; *M*(*SE*) = Mean (Standard Error); *MOE* = Margin of Error.

These findings suggest that higher precision can be achieved by adding more information, i.e., state or contextual variables, to the state-trace or by increasing the model's resolution, i.e., an increased level of binning for the quantization. It should be noted, however, that increasing the model's resolution alone may not always improve the accuracy of the base model's representation if essential information is missing in the model. This is illustrated in Table 4.9, which compares the number of state-trace records obtained from the four different Markov Chain models. As can be seen from the spread measures, the inclusion of the additional check point indicators (MC2) reduces the dispersion of the run lengths of the Markov Chain in both the low and the high-resolution models. However, the higher resolution binning only improves the results of MC2, but does not result in a better estimation of the run length of MC1.

The battlefield model is a terminating simulation and the calculation of the selected performance indicators requires the accumulation of the corresponding variable values over time. Given that the accumulation of these values is sensitive to the terminating condition, the accurate estimation of the performance indicators by the Markov Chains are highly dependent on identifying sufficiently close run-lengths as compared to the base model. In the absence of the representation of time progression in the state-trace data, it is not possible to reliably and accurately represent the base model's behavior even if the model resolution is increased. In fact, findings in Table 4.9 suggest that the additional checkpoint indicators in MC2 are crucial for representing the base model's terminating behavior. Therefore, we will continue the validation with the higher resolution MC2.

Table 4.9. Number of state trace records generated in N=50 runs from different simulation experiments

	Low resolution binning			High resolution binning		
	M	*SD*	Range	*M*	*SD*	Range
Base model [a, b]	655.28	75.76	526-758	655.28	75.76	526-758
MC1 with elapsed time, speed, elevation, movement indicator	640.02	659.40	7-2883	631.16	878.09	8-5305
MC2 with elapsed time, speed, elevation, movement indicator, checkpoint indicators	641.76	127.07	406-1103	656.64	93.55	458-905

Note. [a] Low and high resolution indicate the level of quantization used for the Markov Chains and therefore do not apply to the base model. [b] Spread measures given for the base model belong to the *validation* set.

Table 4.10 presents results of the Student's *t*-test to test the null hypothesis H_0: $M_{Battlefield} = M_{MC2}$ for each of the four performance indicators obtained from model MC2 with higher resolution binning. The findings indicate that there were no significant differences between the mean performance measures calculated from the Battlefield base model and the MC2 Markov Chain.

Table 4.10. Student's *t*-test results for comparing mean performance measures obtained from the Battlefield base model and MC2 with higher resolution binning (HR) for 50 repetitions

	Base model (HR)		MC2 (HR)			
	M	*SE*	*M*	*SE*	|*D*|	*t*(df), *p*
Active mission time section 1	1054.35	3.13	1060.30	8,39	5.95	-0.66(62.37), $p = .509$
Active mission time section 2	906.65	3.89	912.98	6.78	6.33	-0.81(79.04), $p = .421$
Total elevation climbed section 1	406.40	8.01	402.40	8.10	4.00	-0.35(98), $p = .726$
Total elevation climbed section 2	383.80	10.31	397.20	9.31	13.40	-0.97(98), $p = .337$

Another test that we can conduct to validate our Markov Chain results is the two- sample Kolmogorov-Smirnov test. Different from the Student's *t*-test which we used to determine if the difference in means of the performance measures is statistically significant, the two-sample Kolmogorov-Smirnov test evaluates the null hypothesis H_0: the values of a particular KPI obtained from the base model and the higher resolution MC2 come from a population with the same distribution. The findings presented in Table 4.11 indicate that for each four performance measures, we cannot reject the hypothesis that the values for all four KPIs from the higher resolution MC2 and the base model come from the different distributions.

Table 4.11. Two-sample Kolmogorov-Smirnov test results for comparing the distributions obtained from the battlefield model and MC2 with high-resolution binning for 50 repetitions

	|*D*|	df	Kolmogorov-Smirnov Z	*p*-value
Active mission time section 1	0.24	50	1.20	.112
Active mission time section 2	0.24	50	1.20	.112
Total elevation climbed section 1	0.14	50	0.70	.711
Total elevation climbed section 1	0.18	50	0.90	.393

4.3 Conclusions

In this chapter, we studied a battlefield model case to demonstrate the application of our proposed method on a larger, more complex model than the M/M/1 case study model in Chapter 3. The battlefield model is defined using the Dynamic Structure Discrete-event Systems specification (DSDEVS). We provided the details on the implementation of the dynamic coupling between two sections and between the host cells and the sections, and the messaging dynamics between the model components.

Our results show that for this more complex model, our method is able to obtain Markov Chain models that provide estimates of performance indicators with an acceptable level of precision that do not significantly deviate from and follow a similar distribution as the original model's performance measures, as indicated by the Student's *t*-test and the two-sample Kolmogorov-Smirnov test.

In sum, this chapter demonstrated that our method can generate valid abstractions of relatively larger and more complex DEVS models and is capable of adequately estimating their stochastic behavior. We also demonstrated that a higher precision can be achieved by adding more information to the state-trace (e.g., state or contextual variables) or by increasing the model's resolution (e.g., level of binning for the quantization).

However, our result also suggested that increasing the model's resolution alone will not improve the accuracy of the base model's representation when essential information is missing in the model. Therefore, the modelers must identify and include those variables that are essential for generating valid behavior of the Markov Chain that results in correct estimates of the performance indicators. Examples are progress information over time when elapsed time is used in the state trace records, and information about termination conditions.

In the next chapter, we will apply our method to a microscopic traffic simulation model, which has a relatively larger number of model components and a larger state space than the Battlefield DEVS model.

CHAPTER 5

Automated Discrete-event Model Abstraction: Application to Large-Scale Models

5 Automated Discrete-event Model Abstraction: Application to Large-Scale Models

In the previous chapter, we demonstrated that our method can adequately estimate the stochastic behavior of the Battlefield DEVS model. Although the Battlefield model is larger and more complex than the M/M/1 model in Chapter 3, the number of model components (i.e., tank platoon sections) contributing to the dynamic model behavior of the Battlefield system was constant and only two, and the number of state-variables needed to adequately estimate the performance indicators of interest was low. This chapter will demonstrate the step-by-step application of our proposed method to a microscopic traffic simulation model of a road network consisting of a two-lane highway with an on-ramp. Different from the Battlefield case study in Chapter 4, the "short-merge" case study model has a varying number of model components, where vehicles enter and leave the model. The goal of this chapter is twofold: First, test whether our method can be applied to simulation models with relatively larger number of model components and a larger state space. Second, investigate the applicability of the method on a simulation model with a varying number of model components.

5.1 Short-merge Case Study

The short-merge demo model was built as a part of the *OpenTrafficSim*[23] project (Tamminga, 2019; van Lint et al., 2018; van Lint et al., 2016). The model is a microscopic traffic simulation model designed to estimate traffic for a segment of a road network that includes a two-lane highway with an on-ramp (see Figure 5.1). The model investigates the individual vehicle interactions on the highway ramp using the LMRS (Lane change Model with Relaxation and Synchronization) integrated lane change model (Schakel et al., 2012; Schakel, 2015). Similar to the M/M/1 model discussed earlier in §3.2, the short-

[23] OpenTrafficSim, developed at Delft University of Technology, is an open source (micro, macro, meta) traffic simulation framework combining all traffic modes (e.g., cars, buses, pedestrians, airlines) in a single simulator. More information can be found at https://www.opentrafficsim.org/.

merge simulation model describes an open system, with a total number of around 2,500 vehicles, also referred to as *generalized traffic units* or GTUs, generated at random instances[24] of the one-hour simulation run, flowing in and out of the simulated system boundaries. The scenario used in this case study simulates vehicle traffic that only consists of cars and trucks, depicted as blue and red rectangles respectively in the animation (see Figure 5.1).

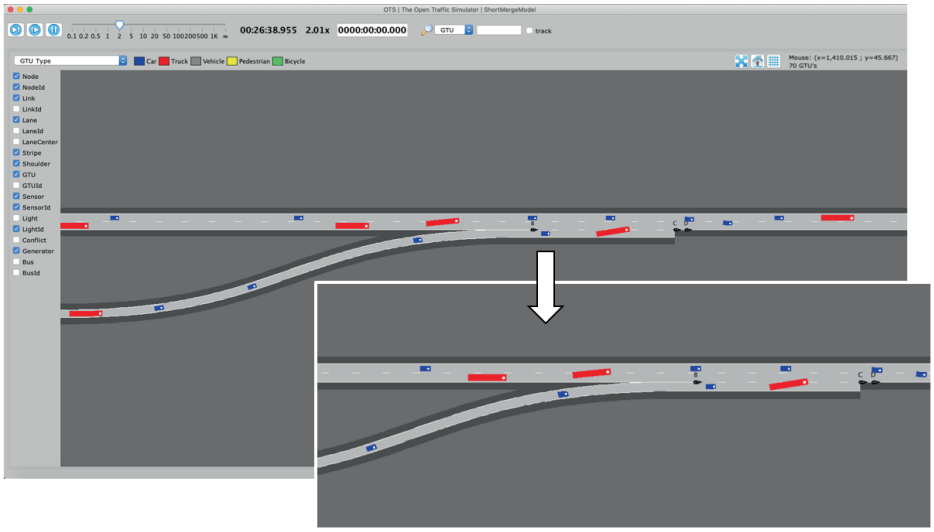

Figure 5.1. The road network and the vehicle traffic as shown in the short-merge animation

Road networks modeled using OpenTrafficSim models are implemented as directed graphs, using nodes and links (Tamminga, 2019; van Lint et al., 2018; van Lint et al., 2016). For example, the nodes of the short-merge model are A, B, C, D, E, F, and F2 (see Figure 5.2), whereas the links are AB, BC, CD, DE, FF2, and F2B (see italic green texts in Figure 5.2).

[24] Using a stream of pseudo random numbers generated using the Mersenne Twister PRNG with unique incrementing seed values for each simulation run.

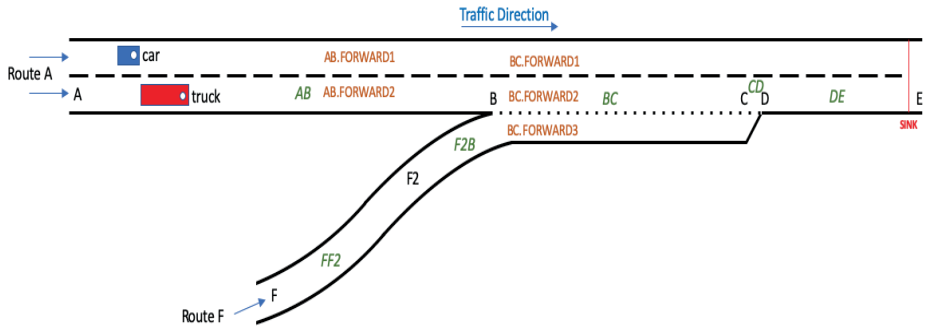

Figure 5.2. Short-merge model components

In the model, the origin/destination split in the network is defined as two routes: Route A (nodes A, B, C, D and E) and Route F (nodes F, F2, B, C, D and E). Routes consists of a list of Nodes. The road network of the short-merge model consists of a two-lane highway (links AB, BC, CD, and DE) and the on-ramp (links FF2 and F2B). Finally, links in the network can be further divided laterally to form *lanes*. This allows modeler to define network demand and traffic flows at the lane level (e.g., link BC is divided into lanes BC.FORWARD1, BC.FORWARD2 and BC.FORWARD3). The number of vehicles to be randomly generated in the short-merge model is set for each lane of the highway (i.e., 1,000 vehicles per lane per hour) and for the on-ramp (i.e., 500 vehicles per hour), separately. The distribution of trucks and cars for a given network demand is also defined per lane per route. This percentage is same for each lane and it is 30% trucks and 70% cars of the total network demand per lane.

Except for the vehicle lengths l, accelerations a, and desired speed, most model parameters are the same between the passenger cars and trucks. In the model, car length l_{car} is defined as 4 m and the truck length l_{truck} is set to 15m. The desired acceleration of the cars a_{car} is 1.25m/s^2 and for the trucks a_{truck} is set to 0.8 m/s^2. The desired speed calculations for both cars and trucks are taken from work by Schakel et al. (2012): the desired speed of a car is calculated based on the driver preference $\delta_{car} = N(v_{des,car}, \sigma_{car})/v_{lim}$ where $N(v_{des,car}, \sigma_{car})$ is Gaussian distribution with mean $v_{des,car}$ = 123.7

km/h, and the standard deviation σ_{car} = 12.0 km/h. The speed limit for cars on route A (i.e., two-lane highway) is 120 km/h and for trucks it is 90 km/h. The speed limit at the entrance of the route F until the F2 node is 20 km/h for both cars and trucks. Once the vehicles reached the node F2 (see Figure 5.2), then the speed limit for cars becomes 120 km/h and for trucks 90 km/h.

For trucks, the calculation of the desired speed is based on the driver preference for the maximum vehicle speed, which is $v_{max,truck} = N(v_{des,truck}, \sigma_{truck})$ where $N(v_{des,truck}, \sigma_{truck})$ is Gaussian distribution with mean $v_{des,car}$ = 85.0 km/h and the standard deviation σ_{truck} = 2.5 km/h. Other parameters of the models are maximum deceleration is b = 2.09 m/s^2, stopping distance s_0 = 3m, maximum headway T_{max} = 1.2s, and minimum headway T_{min} = 0.56s (see Schakel, 2015 for the complete list of LMRS parameters).

It is important to note that the short-merge model implementation includes several additional behavioral aspects such as observing traffic, lookahead and lookback for lane determination, relaxation, and synchronization, which makes the behavior of each car and truck, and the interactions between vehicles more complete. These additional behavioral aspects, however, increase model complexity and causes model to run slow. This makes the short-merge model a suitable candidate for our method.

5.2 Application of the Temporal Data Mining-based Model Abstraction Method to the Short-Merge Model

In the following subsections, we will illustrate how we applied the method step-by-step to the short-merge model in the same order as presented in Chapter 3.

5.2.1 Generation of Discrete-event Model State-Trace Data

Remember from §3.2.1 that there are six major factors to consider when generating state-traces from a simulation model. Below, we will discuss the decisions for each short-merge model consideration.

5.2.1.1 Representation of Time

The model uses an event scheduling simulator implementation, which updates the states of some of the model components at every event occurrence, for instance, the generation of cars and recalculation of the vehicles' states at a fixed interval τ of 0.5 seconds. In this case study, we are interested in estimating the driving behavior of vehicles in the base model and the effects of the merge to the state variables such as the vehicle speed, vehicle acceleration, and traffic intensity over time. Therefore, it is essential for us to accurately capture the order of changes in the vehicles' states and exact times of these changes throughout a run. Consequently, we choose an interval of 0.5 seconds (i.e., the fixed interval that the simulator recalculates each vehicle's state, where the vehicles do not recalculate their new state at the same time) for the state-trace reporter to log new state-trace records containing time, system states and several additional parameters (e.g., input/output parameters and contextual variables) for each run. A smaller interval would result in the repetition of the same vehicle states and a larger state-trace data set. Meanwhile, a larger interval may not contain essential updates in the vehicle states, the generation of new vehicles at Nodes A and F, the deletion of vehicles at Node E, and the changes in the traffic intensity, which may affect the accuracy of the Markov Chain estimates.

To report the state of the system that contains the simulation time information at a fixed time-step of 0.5 seconds, we schedule (and reschedule) an event (i.e., a call of the "reportState" method) with a relative duration of 0.5 seconds in simulator's event queue until the total simulation duration. This way, the method call is executed by the simulator at every (*simulator time* + *relative duration*). This is with the exception of the reporting of the initial state of the system with the simulation time and other additional parameters, which is reported at time 0. The reporting of the final state of the system with the simulation time and other additional parameters is done at time 3,599.5s.

However, because the step-size is constant at 0.5 seconds, the inclusion of the fixed time step size in the state-trace data as a separate column will not affect the variability of the episodes discovered by the episode mining algorithm. Therefore, we excluded the simulation time from the state-trace data. When calculating the mean performance indicators

from the Markov Chain generated state-trace data, we will assume that each row in the data corresponds to a time progression of 0.5 seconds.

5.2.1.2 Stochastic vs. Deterministic Discrete-event Models

The short-merge model is stochastic. The model uses several random variables with their own distributions; for example, to generate vehicles and for the lane changing behavior (Schakel et al., 2012). The vehicles flow in to the road network with a randomly generated initial speed at a rate defined by the network demand per lane per vehicle type. This is implemented by creating a stream of generation times for vehicles using a pseudo-RNG. The seed management for the RNG ensures that each run has a unique traffic pattern while being fully reproducible. For the state-trace data generation, 100 runs were used with each run having a different single incrementing seed value (e.g., [seed = 1, run = 1], [seed = 2, run = 2], …, [seed = 100, run = 100]). This data is split into two equal-sized non-overlapping training and validation sets. The training data will be used for the episode mining and the validation data will be used for the model validation stage (see Table 5.1).

Table 5.1. Seed value ranges of the RNG for each dataset in the case study

Model	Data set	# of state traces per dataset	Seed value range
Short-merge (base model)	Training	50	1-50
	Validation	50	51-100

5.2.1.3 Terminating vs. Non-terminating Simulations

The short-merge simulation is non-terminating. The simulation has no external event that determines the end of a run and it would run continuously in the absence of an internal event scheduled at a predetermined absolute time to terminate the run. Therefore, we scheduled a method call to be executed just before the simulation clock reaches to the predetermined end time of the simulation (i.e., 3,600 seconds) to close the stream and release the system resources associated with it.

There is no warm-up period before starting the state-trace data collection[25]. Each simulation run starts with the same initial condition of no-vehicle traffic on the network. Cars and trucks are only generated at the start of route A and F, and a vehicle cannot have a starting location other than these locations in the network.

Each state-trace record generated from the runs has the same number of state-trace records, i.e., 7,200. Given that the simulation starts with the same initial condition, terminates after the same predetermined run-length, and the reporting of each state-trace record is done after every 0.5 seconds passed, each state-trace record generated from each run contains the same number of 7,200 state-trace records, including the reporting of the initial condition and other parameters at $t = 0$ and the reporting of the final state at $t =$ 3,599.5.

It is important to note that, although the simulation is non-terminating, each vehicle atomic model component is designed to exhibit a terminating behavior. That is, a vehicle enters the system after being generated by the generator and leaves the system once it is removed at the departure Node E. Therefore, the actual states belong to a vehicle are reported only during the time they are active in the model. When the vehicles are inactive either due to not being generated yet or already removed from the system, all state variables (with the exception of the state-variable *active*) are represented as "NA" in the data. This ensures that all records have the same number of columns, consistent with the data sets of chapters 3 and 4.

5.2.1.4 Inclusion of Input Data

The short-merge case study investigates a single scenario with a single (fixed) set of input parameters and no external input. Some of these input parameters are related to the underlying lane-change (LMRS) model and an overview of these input parameters can be

[25] Although the inclusion of the warm-up period in the experiment should not change the application of the proposed method, further research is needed to explore the accuracy of the method and the resulting Markov Chain in terms of capturing the transient behavior – in this case, it is the build-up of the traffic – of the non-terminating base model.

found in (Schakel et al., 2012). Other input parameters such as the lane demands, fraction of passenger cars and trucks, and the length of a run and their values used in our experiments are given in below Table 5.2.

Table 5.2. Input parameters of the short-merge model

Input parameter	Value
Passenger car Fraction	70%
Truck Fraction	30%
Main road demand	2,000 per hour
On-ramp demand	500 per hour
Simulation duration	3,600 seconds

Given that the input parameters are fixed across all runs and the values remain constant due to the lack of external inputs, the inclusion of input parameters in the state-trace data will not affect the variability of the episodes discovered by the episode mining algorithm. Therefore, the input parameters of the model are excluded from the state-trace data.

5.2.1.5 Inclusion of Output Data

The short-merge model estimates the key performance indicators (KPIs) shown in Table 5.3.

To estimate these performance indicators using the state-trace data, we include the key variables (6) listed in Table 5.4 for each vehicle (2,500) as individual columns in the state-trace data:

Table 5.3. Key performance indicators of the short-merge model

Mean performance indicator[26]	Description	Example Value[27]
avgSpeedACar_Mean	The average speed of the cars that left the model and started at the left side of the road network (Route A)	24.53 km/h
avgSpeedFCar_Mean	The average speed of the cars that left the model and started at the bottom side of the road network (Route F)	19.53 km/h
avgSpeedATruck_Mean	The average speed of the trucks that left the model and started at the left side of the road network (Route A)	18.52 km/h
avgSpeedFTruck_Mean	The average speed of the trucks that left the model and started at the bottom side of the road network (Route F)	17.04 km/h
modelTimeACar_Mean	Average time in the system for cars that left the model and started at the left side of the road network (Route A)	133.65 s
modelTimeFCar_Mean	Average time in the system for cars that left the model and started at the bottom side of the road network (Route F)	87.75 s
modelTimeATruck_Mean	Average time in the system for trucks that left the model and started at the left side of the road network (Route A)	173.75 s
modelTimeFTruck_Mean	Average time in the system for trucks that left the model and started at the bottom side of the road network (Route F)	99.47 s

[26] The names of the mean performance indicators given in Table 5.3 are the names of the run statistics variables used in the model.

[27] The example values shown in Table 5.3 are from the state-trace data generated from simulation run #1.

Table 5.4. The subset of short-merge state variables included in the state-trace data, with an example value for vehicle number 1

VariableName_VehicleId	Description	Example Value
active_1	0 = inactive, 1 = active	1
carTruck_1	car or truck; car = 0; truck = 1	1
laneId_1	lane of the vehicle	AB.FORWARD2
speed_1	speed in km/h	64.00
acceleration_1	acceleration in m/s^2	0.14
distanceHalfSec_1	driven distance in current half second	12.52 m

Note that the performance indicators of the short-merge model are calculated per route per vehicle type. Therefore, to estimate these indicators from the state-trace data, we should first identify all vehicles with the same vehicle type and the same starting side of the network using the *carTruck* and *laneId* variables, respectively. Additionally, the run statistics are calculated only for vehicles that have exited the system. Therefore, to determine if a vehicle left the model before the end of a simulation, we use the vehicles' *active* variables. When a vehicle is generated, the variable is updated from 0 to 1, and it is reset to 0 when it is removed at Node E. As a result, any vehicle that has an *active* variable transition from 1 to 0 is assumed to have exited the boundaries of the modeled system, and only those vehicles will be included in the statistics calculation. The initial *laneId* of a vehicle is later retrieved from the state-trace data by looking at the first row where *active* is 1.

The subsequent sub-sections will explain how we will estimate of the base model's performance indicators shown in Table 5.3 using the variables included in the state-trace data generated by the Markov Chain. It should be noted that the same calculation methods could also be used on the initial state trace data set from the base model.

5.2.1.5.1 Calculation of Average Time-in-system Using State-Trace Data

To calculate the average time-in-system from the state-trace data:

(I) For the passenger cars that have left the model and started at the left side of the road network (Route A), store the sum of the *time-in-system* for all vehicles with *carTruck* equals to 0 and *laneId* equals to *AB.FORWARD1* or *AB.FORWARD2* that left the system boundaries (i.e., the active variable value transitions from 1 to 0). Time-in-system for a single vehicle is calculated by multiplying the number of rows where the *active* variable is 1 with the time interval $\tau = 0.5$s.

(II) To calculate the average time-in-system for all passanger cars generated at the left side of the road network and left the system, we first sum their time-in-system values. Then, we divide this sum by the count of Route A passenger cars that have left the model.

(III) To calculate the average time-in-system for trucks, or for vehicles started at the bottom of the road network that left the system, we adjust the conditions for the *carTruck* and *laneId* values accordingly.

5.2.1.5.2 Calculation of Average Speed Using State-Trace Data

To calculate the average speed from the state-trace data:

(I) For the trucks that have left the model and started at the bottom side of the road network (route F), the average speed of an individual truck is the sum of its non-N/A distanceHalfSec (i.e., total distance traveled) values divided by the time-in-system of that specific truck. In §5.2.1.6, we will elaborate on the distinction between mining state-trace data at the atomic model (vehicle) level and mining state-trace data at the system level.

(II) To calculate the average speed for all trucks that left the model and started at the bottom side of the road network (route F), we first sum their individually calculated average speed values. Then, we divide the sum by the count of Route F trucks that left the model.

(III) To calculate the average speed for cars or for vehicles that started at the left side of the road network that left the system, we adjust the conditions for the *carTruck* and *laneId* values accordingly.

5.2.1.6 Complete vs. Partial Model State-Trace Data

For this case study, we use partial state-trace data containing a subset of the state variables available from the vehicle atomic model (see Appendix C for the complete list of the variables, example values, and their descriptions). Our goal is to estimate the performance indicators of interest given in Table 5.3 from the Markov Chain, which is generated by applying our method to the base model's state-trace data resulting in an abstraction of the original short-merge model. We hypothesize that we may achieve a sufficiently enough estimation of these mean performance indicators using the partial model state-trace containing the variables listed in Table 5.4.

Recall from Chapter 3 and 4 that for M/M/1 and Battlefield case studies, we applied the frequent episode mining algorithm to the state-trace data, where each state-trace record is reported as a new row at each event occurrence or at fixed time intervals, and it represents the total state of the system at the reported instance. Therefore, we encoded each unique row containing all the desired variables to a single positive integer value (see Figure 5.3.a). A particular advantage of the encoding of all data in a row is that the information regarding the states of individual model components relative to the others in the system is preserved. The number of atomic model components (i.e., 2,500 vehicles) in the short-merge model, on the other hand, is significantly greater than in the M/M/1 and battlefield models, and a single row, having more than 17,000 columns, can take one of 45 million unique values. Because of the large number of possible unique states, the frequent episode mining algorithm with the input parameters of *maxwin* of 2 and a *support* of 1 may generate a very large n for a $n \times n$ transition probability matrix, and thus a huge Markov Chain.

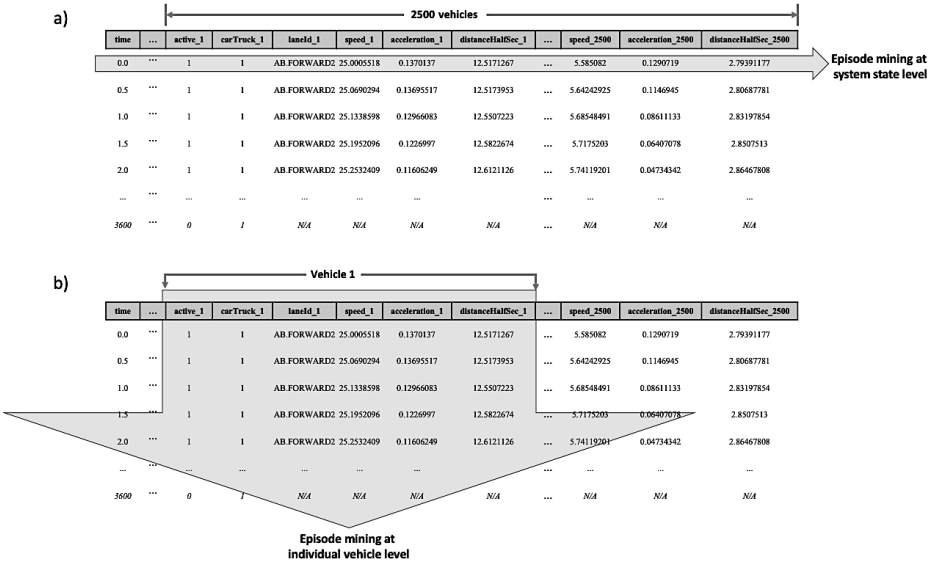

Figure 5.3. System state and model component state level episode mining

Instead of encoding state-trace records in each row to a single value representing the system state, a state-trace record can be deconstructed into multiple state traces where each trace represents the state-progression of individual (atomic or coupled) model components (as shown in Figure 5.3.b). Then, the frequent episode mining algorithms can be applied to these component-based traces to generate component-based Markov Chains. However, mining at component level, or individual vehicle level, may omit contextual insight into how components interact with and affect each other, i.e., the behavior of the system as a whole. We will elaborate more on this assumption in the §5.2.1.6.1.

To test our assumptions, we designed two experiments to assess the effects of (1) system level abstraction using partial state using partial model state-trace without contextual variables (ME1), and (2) component-level abstraction with the additional contextual variables included in the state-trace data (ME2). For both experiments, we created two datasets, each containing the same amount of state-traces (100) with an equal split of training (50) and validation (50) data. For the first experiment (ME1), we used the partial model state-trace containing the variables listed in Table 5.4. For the second experiment (ME2), we introduced a set of contextual variables (as shown in Table 5.6), indicating the number of vehicles per link and lane at every fixed time interval of 0.5 seconds.

5.2.1.6.1 Inclusion of the Traffic Intensity Contextual Variables for Experiment ME2

In the short-merge model, the speed and acceleration of each vehicle is affected by the behavior of other vehicles around them. Without the inclusion of the traffic information in the component-level Markov Chains, a valid representation of the driving behavior of vehicles, and thus, accurate estimation of the base model's performance indicators from the simulation of individual vehicle-based Markov Chains may not be possible. Therefore, the traffic intensity contextual variables in Table 5.4 are included in the component-level Markov Chain state-traces to re-capture the missing information of vehicle traffic on each link and lane.

We hypothesize, based on the fact that there is a well-known correlation between the overall intensity of the traffic on a stretch of road and the average speed of vehicles (Wong et al., 2021), that the combination of these variables and the aforementioned state-variables used in ME1 for each individual vehicle atomic component would suffice to the relationship between (average) vehicle speed and traffic intensity and estimate the mean performance indicators of the system.

Table 5.6. Contextual variable set for Experiment ME2- Number of vehicles per link per lane at a certain time instance

Variable-Name_laneId	Description	Example Value	Value range
nrVehiclesAB_d1	These contextual variables indicate the number of cars on each link at that particular instance of time. For example, the first variable nrVehicleAB_d1 indicates that the number of vehicles on the first lane of the link AB at that particular instance was 30. Similarly, the second variable nrVehiclesAB_d2 indicates that the number of vehicles on the second lane of the link AB at a particular instance was 67.	30	[0-80]
nrVehiclesAB_d2		67	[0-132]
nrVehiclesBC_d1		1	[0-6]
nrVehiclesBC_d2		3	[0-8]
nrVehiclesBC_d3		0	[0-6]
nrVehiclesCD_d1		1	[0-1]
nrVehiclesCD_d2		0	[0-2]
nrVehiclesDE_d1		21	[0-33]
nrVehiclesDE_d2		27	[0-41]
nrVehiclesF2B_d1		1	[0-20]

5.2.2 Application of the Temporal Data Mining Tasks to the State-Trace Data

In this section, we will examine the specific challenges for the short-merge state-traces and our choices for the preprocessing of the state variables. Then, we will illustrate how the frequent episode mining algorithm is used to calculate the transition probabilities from the state-traces and build the Markov Chain's transition probability matrix.

5.2.2.1 Preprocessing of the State-Trace Data

Following the completion of state-trace generations, the next step in our method is the preparation of the data for the episode mining algorithm. Among the variables listed in Figure 5.4, the continuous variables *speed*, *acceleration* and *distanceHalfSec* are quantized using an equal-width binning based method. We chose the bin widths 10, 5, and 5 for the speed, acceleration and distanceHalfSec variables, respectively. These bin-widths are chosen based on the value ranges of the variables with the goal of reducing the effect of over-population of candidate episodes with low frequency values.

The categorical variables *active*, *carTruck* and *laneId* are not quantized. The variables *active* and *carTruck* have a limited number of possible states, that is, both variables are coded with binary values 0 and 1, and the carTruck variable is additionally represented as NA during to the inactive period of vehicles. The *laneId* categorical variable can takes twelve different values (including the value "NA"). These values include the "AB.FORWARD1", "AB.FORWARD2", "FF2.FORWARD1" values, which are previously mentioned in §5.2.1.5.1 and §5.1.2.5.2 and important to identify the generation locations for the vehicles and accurately calculate the performance indicators in Table 5.3. Although the removal of lane information from the state-trace data would result in losing the information on the lane changes, overtaking, and the relation between fast driving and the "left lane", the KPI calculations can still be done.

For the ME2 experiment, we quantized the discrete traffic intensity contextual variables as their value range could lead to a large number of unique episodes with low frequencies. For the quantization of the traffic intensity contextual variables, we used equal-width binning strategy based on the ranges obtained across all 50 runs (see Table 5.6 for the range values). Given that each deconstructed component-level trace has the full set (10) of traffic intensity contextual variables in addition to the state variables, we wanted to

limit the number of bins to less than ten (and adjusted the numbers according to each variable's value range) to limit the generation of candidate episodes with low frequencies.

As the final step of the preprocessing step for this short-merge case study, we encode state-trace records containing the preprocessed variables for the frequent episode mining algorithm. The training set generated for ME1 contains 50 state traces, each with 7,200 rows and 15,000 columns (i.e., 2,500 vehicles with 6 state variables, listed in Table 5.4). Because we intend to perform system-state level episode mining on the ME1 training set, the data is encoded horizontally at the state-trace record level. However, the encoding process for the ME1 training set revealed already from the first eight state trace files that 64,512 of 64,000 (i.e., 99.56%) state-trace records had a frequency of 1. Unfortunately, a data set with such a low level of repetition (note that the data was already quantized) would not be suitable for the frequent episode mining task since there are no frequent episodes. The Markov Chain model would therefore, in essence, just replay one of the 50 runs of the base model. As a result, we decided to exclude ME1 from further analysis. An important lesson is that the more variables are included in the state trace data set, the lower the chance to find frequent episodes. Case ME1 with 15,000 state variables was a relatively extreme example, but it clearly illustrates this problem.

5.2.2.1.1 Subdividing the State-Trace for Experiment ME2

One more adjustment we make for Experiment ME2 before the application of the frequent episode mining algorithm is subdividing the tabular state-trace data per vehicle as shown in Figure 5.3.b. The data generated by the trace-reporter method in the short-merge model contains the states of all vehicles in the same row (i.e., a system-level state-trace record), horizontally sorted by the vehicle id. Because the strategy to test in Experiment ME2 is to construct component-based, or vehicle-based, Markov Chains for each individual vehicle in the system, the data must be represented in a format for the EMMA algorithm containing only the states belonging to the same vehicle as an input for the calculation of the transition probabilities, and eventually, to generate the transition probability matrices per vehicle.

Although the EMMA algorithm can process multiple positive integer values presented in a single row, where each value is separated by a single space, it requires these multiple

values to be sorted according to a total ordering (e.g., ascending) and with no repetition of the same value in the same row (Huang & Chang, 2008). Therefore, we subdivided each row in the state-trace data for each of the 2,500 vehicles, ordered sequentially based on their gtuIds[28]. Based on section 5.2.1.6.1, we added the contextual variables for the traffic intensity per link (Experiment ME2) to each of the 2,500 subdivided state trace data sets, so each row contains the combined information of the state of a particular vehicle and the values of the link-based or lane-based traffic intensity variables, encoded as a single positive integer value. The final version of a fully-subdivided state-trace data set contains 18 million rows per run (2,500 vehicles with 7,200 rows per vehicle). This data is either stored in a single file with 18 million records, where each 2,500th record contains data about the same vehicle, or in 2,500 separate vehicle files.

5.2.2.2 Application of the Frequent Episode Mining Algorithm to Preprocessed State-Trace Data

Following the preprocessing of variables in the data set, the EMMA algorithm is applied to the training data. The two input parameters of the algorithm *minsup* = 1 and *maxwin* = 2 are the same as the values used in Chapter 3 and 4.

For Experiment ME2, however, we introduced a "skip factor" variable with a value of 2,500 for the algorithm to process the data based on the changes explained in the previous section when all 18 million records are stored in a single file. Based on the combination of values of the new and the standard input parameters, the algorithm discovers frequent episodes with the minimum support of 1 with window length 1 (i.e., vehicle states + link-based or lane-based traffic intensity values) and window length 2 (i.e., two consecutive rows of the same vehicle 0.5 seconds apart + link or lane traffic intensity values 0.5 seconds apart), and outputs the transition probability matrices for each of the 2,500 vehicles.

---

[28] In OpenTrafficSim, the term GTU stands for "generic traffic unit" and represents a person or a vehicle identified with a unique "gtuId" number. In short-merge model, all vehicles are assigned an incremental gtuId number from 1 to 2,500.

5.2.3 Simulation of the Discrete-time Markov Chains

For Experiment ME2, the frequent episode mining algorithm was applied to the training set to discover frequent states, frequent state-transitions, and state-transition probabilities for each of the 2,500 vehicles. After this, the next step in our method is to perform random walks for 2,500 individual vehicle-based Markov Chains according to their transition probability matrices. To obtain state-trace data for the validation study, we performed an experiment with 50 repetitions, where each repetition simulates all 2,500 vehicle-based Markov Chains with the same pseudo-RNG seed value for each run and a different seed for each repetition. The experiment generated a total of 125,000 state-trace data sets (i.e., 50 runs × 2,500 state-traces). In the validation stage of our method, we will calculate the aggregated mean performance measures using the results obtained from all vehicle-based Markov Chains and compare the estimates of these performance measures with the base model's performance measures.

5.2.4 Validation of the Markov Chain Results

5.2.4.1 Validation of the results from experiments ME2

To validate the results of the Markov Chain (ME2), which is t models subdivided per vehicle, we compare the performance of the base model and the Markov Chain on three different measures: (a) the number and distribution of vehicles generated per route; (b) average speed per route per vehicle type (KPI); and c) average time in the system per route per vehicle type (KPI). Table 5.7 presents the descriptive statistics of these measures for the base model and the Markov Chain over 50 runs. As this table shows, the average numbers of vehicles contributing to the KPI calculation in the Markov Chain (ME2) are lower than numbers of vehicles in the base model. This may be explained by the difference between the arrival processes of the base model and the Markov Chain (ME2). Given a sequence of n interarrivals times $X_1, X_2, \cdots, X_n$ for the arrival epochs S_1, S_2, …, S_n, the arrival time of the n^{th} epoch S_n for a given X_n is calculated by the joint distribution of $X_1, X_2, \cdots, X_n$ for all n>1, and specified as

$$S_n = \sum_{i=1}^{n} X_i \tag{5.1}$$

As indicated by the Central Limit Theorem, these sum of $X_1+X_2 + \cdots + X_n$ for a large n, where the X_1, X_2, ⋯, X_n are drawn from any distribution with mean μ and standard deviation σ, converges to a Normal distribution with mean $n\mu$ and standard deviation $\sigma\sqrt{n}$. The distribution, therefore, narrows when n gets larger as the mean μ scales by n and the standard deviation scales by $\sqrt{n}$. In the case where the average interarrival time β, the arrival time of the given X_n (i.e., nth vehicle) is the sum of n independent drawings from the Exponential distribution, with average interarrival time β is an Erlang distribution (Li & Li, 2019):

$$\text{if } X_i \sim \text{Exponential}(\beta), \text{then } \sum_{i=1}^{n} X_i \sim \text{Erlang}(n, \beta) \tag{5.2}$$

which has a mean of $n\beta$ and a standard deviation of $\beta\sqrt{n}$, consistent with the Central Limit Theorem. When the generators in the short-merge model generate a new vehicle at every 2 seconds, the 1,000th vehicle is generated on average at time t = 2*1,000 = 2,000 seconds. The standard deviation is, therefore, $2\sqrt{1000}$ = 63. This generates a small sigma value: 95% of the arrival times will be between μ +/- 2.5 σ = time interval [1,842-2,158].

In the Markov Chain, the interarrival times follows a memoryless geometrical distribution, where X_1, X_2, ⋯, X_n are iid Bernoulli variables. The geometric distribution represents the probability of getting the first occurrence of X_n after n consecutive independent trials:

$$P(X = n) = (1 - p)^{n-1} p \qquad n = 1,2,3 \ldots \tag{5.3}$$

where p is the success probability of each trial and the expected value of the distribution is

$$E(X) = \frac{1}{P} \tag{5.4}$$

For the generation of a vehicle at time t seconds, the Markov Chain requires $2t$ draws (0.5 seconds sample rate x 2,000 state-trace records). Meaning that for a vehicle that is expected be generated around 2,000 seconds, the geometric distribution should be calculated for p = 1/4,000. The mean of the geometric distribution is 1/p, indicating that, on average, the vehicle is generated at state transition 4,000, corresponding to t = 2,000.

The standard deviation of the Geometric distribution is $\sqrt{\frac{1-p}{p^2}}$, which is 3,999.5 for p = 1/4,000. This is a huge sigma value: based on the Exponential (continuous) analogue of the geometric distribution, 95% of the arrival times would be between [0-6,000], which is much higher than the base model's [1,842-2,158] confidence interval. Using the Exponential approximation of the Geometric distribution, we can calculate that there is a 16% chance that a vehicle that should be generated at t = 2,000 will not be generated in the runtime of the Markov Chain experiments. Especially for the vehicles with higher gtuIds in Markov Chain (ME2), the spread of the geometric distribution will get significantly larger than the Erlang distribution, with a long right tail (i.e., right-skewed) and the Markov Chain (ME2) will not generate vehicles where the calculated value is higher than 3,600 seconds (or 7,200 state transitions), since the run ends at t = 3,600 seconds. Despite the lower absolute numbers of vehicles in the Markov Chain (ME2), the Route A to Route F ratio of vehicles (in bold) is nearly identical between both models. Moreover, within each route, the cars to trucks ratio – which was an input parameter of the base model – was accurately reconstructed by the Markov Chain (ME2).

Table 5.7. Descriptive statistics for measures obtained from the short merge base model and the Markov Chain for Experiment ME2

	Base model		Markov Chain (ME2)	
	M (*SD*)	% or Range	*M* (*SD*)	% or Range
Number of generated vehicles used for KPI calculation per route per vehicle type				
Route A Total	**1,908.76 (51.17)**	**79.80**	**1,201.06 (28.59)**	**79.93**
Route A Cars	1,621.44 (47.51)	84.95	1,025.54 (24.21)	85.39
Route A Trucks	287.32 (18.32)	15.05	175.52 (15.07)	14.61
Route F Total	**483.02 (22.15)**	**20.20**	**301.60 (16.36)**	**20.07**
Route A Cars	335.78 (18.67)	69.52	210.92 (14.04)	69.93
Route A Trucks	147.24 (12.23)	30.48	90.68 (10.73)	30.07
Average speed per route per vehicle type				
Route A Cars	22.82 (2.69)	15.07-26.89	23.28 (0.24)	22.81-23.68
Route A Trucks	17.85 (2.39)	11.98-21.94	18.49 (0.40)	17.53-19.24
Route F Cars	19.18 (0.54)	18.15-20.35	18.14 (0.30)	17.36-18.79
Route F Trucks	17.72 (0.37)	16.71-18.44	16.75 (0.36)	16.01-17.63
Average time-in-system per route per vehicle type				
Route A Cars	151.79 (30.13)	116.12-264.88	145.95 (3.56)	139.15-157.07
Route A Trucks	203.38 (44.05)	148.24-349.40	192.10 (10.32)	167.03-210.72
Route F Cars	91.88 (3.11)	86.02-99.21	91.59 (3.58)	85.80-99.48
Route F Trucks	101.54 (2.86)	97.02-110.64	100.57 (4.28)	92.98-110.79

Table 5.8 presents results of the Student's *t*-test to test the null hypothesis H_0: $M_{Short\_merge} = M_{MC\_ME2}$ for each of the eight performance measures obtained from the ME2 runs. For the average speed KPI, the findings indicate that there were no significant differences between both Route A mean speed measures calculated from the short merge base model and the Markov Chain (ME2). However, for Route F, the car and truck mean speeds generated from the Markov Chain (ME2) differed significantly from the mean speeds obtained from the base model. This may be explained by the fact that route F contains fewer vehicles, and therefore, less data to train the Markov Chain (ME2). Even though the means are significantly different from each other, the magnitude of the difference is small, around 1 m/s.

For the average time-in-system KPI, all *t*-test were non-significant, indicating that the mean times that vehicles travelled within the boundaries of the system did not differ significantly between the base model and the Markov Chain (ME2).

Table 5.8. Student's *t*-test results for comparing mean performance measures obtained from the short merge model and the Markov Chain for 50 repetitions for Experiment ME2

	Base model		Markov Chain (ME2)			
	M	*SE*	*M*	*SE*	|*D*|	*t*(df), *p*
Average speed per route per vehicle type						
Route A Cars	22.82	0.38	23.28	0.03	0.46	-1.21(49.78), $p = .232$
Route A Trucks	17.85	0.34	18.49	0.06	0.63	-1.85(51.76), $p = .070$
Route F Cars	19.18	0.08	18.14	0.04	1.04	11.91(76.73), $p < .001$
Route F Trucks	17.72	0.05	16.75	0.05	0.97	13.37(98), $p < .001$
Average time-in-system per route per vehicle type						
Route A Cars	151.79	4.26	145.95	0.50	5.84	1.36(50.37), $p = .180$
Route A Trucks	203.38	6.23	192.10	1.46	11.28	1.76(54.37), $p = .083$
Route F Cars	91.88	0.44	91.59	0.51	0.28	0.42(98), $p = .675$
Route F Trucks	101.54	0.40	100.57	0.60	0.98	1.34(85.47), $p = .183$

Using the two-sample Kolmogorov-Smirnov test, we also evaluated the null hypothesis H_0: the values of a particular KPI obtained from the base model and the Markov Chain (ME2) come from a population with the same distribution. The findings presented in Table 5.9 show that most of the Kolmogorov-Smirnov tests indicated significant

differences between the distributions of the base model and those of the Markov Chain (ME2). Only for Route F cars' average time in system, results suggest that the values obtained from the base model and the Markov Chain (ME2) come from a population with the same distribution. The finding that the distributions of the base model and Markov Chain (ME2) key performance measures differed can be explained by observing the descriptive statistics presented in Table 5.7. For the Route A vehicles, the spread (as indicated by the standard deviation and the range) of the Markov Chain (ME2) average speed estimates appears consistently narrower compared to the spread of the base model estimates. Given that Route A has fewer congestions in general than Route F, it is possible that the relatively rare samples of congested traffic were underrepresented in the Markov Chain (ME2). This would result in a distribution more centered around the mean. For Route F, descriptive statistics in Table 5.7 show that the base model and the Markov Chain (ME2) estimates have a similar spread. The significant K-S test result for speed could be driven by the small but statistically significant mean difference identified in the *t*-test.

Table 5.9. Two-sample Kolmogorov-Smirnov test results for comparing distributions obtained from the short merge base model and the Markov Chain for 50 repetitions for Experiment ME2

	$\lvert D \rvert$	df	Kolmogorov-Smirnov Z	*p*-value
Average speed per route per vehicle type				
Route A Cars	.44	50	2.20	<.001
Route A Trucks	.44	50	2.20	<.001
Route F Cars	.80	50	4.00	<.001
Route F Trucks	.84	50	4.20	<.001
Average time-in-system per route per vehicle type				
Route A Cars	.44	50	2.20	<.001
Route A Trucks	.38	50	1.90	.002
Route F Cars	.16	50	0.80	.544
Route F Trucks	.28	50	1.40	.040

5.2.4.2 Tackling the issue of low vehicle numbers in Markov Chain (ME2)

To address the issue of the low number of vehicles in the Markov Chain (ME2), we carried out an experiment ME3, where we adjusted the Markov Chain (ME2) by

introducing a *progressIndicator* variable in the state-trace data, which operates as a mechanism to link vehicles' driving behaviour and network link capacities to the time progression of the simulation. We hypothesize that the inclusion of a *progressIndicator* variable may be necessary to correctly distinguish between the active and inactive times of vehicles over the course of a run. The continuous *progressIndicator* variable takes reel values between 0 meter and 1,500 meters, and therefore, has a large value range. We used equal-width binning strategy for the quantization of the values and chose bin-width size 10 to try to limit the low frequency candidate episodes. We then repeated the experiment and the validation steps with the adjusted Markov Chain (ME3) to test whether a better representation of the number of vehicles resulted in improved the KPI estimates. Table 5.10 presents the descriptive statistics for the base model and the adjusted Markov Chain (ME3) over 50 runs. As this table shows, compared to Markov Chain (ME2), the number of vehicles in the adjusted Markov Chain (ME3) increased and became more similar to the number of vehicles in the base model.

Table 5.10. Descriptive statistics for measures obtained from the short merge base model and the adjusted Markov Chain with time progression variable for Experiment ME3

	Base Model		Adjusted Markov Chain (ME3)	
	M (*SD*)	% or Range	*M* (*SD*)	% or Range
Number of generated vehicles used for KPI calculation per route per vehicle type				
Route A Total	**1908.76 (51.17)**	**79.80**	**1379.48 (178.79)**	**79.76**
Route A Cars	1621.44 (47.51)	84.95	1177.00 (154.97)	85.32
Route A Trucks	287.32 (18.32)	15.05	202.48 (33.62)	14.68
Route F Total	**483.02 (22.15)**	**20.20**	**359.20 (48.16)**	**20.24**
Route F Cars	335.78 (18.67)	69.52	241.16 (27.53)	68.87
Route F Trucks	147.24 (12.23)	30.48	109.00 (17.51)	31.13
Average speed per route per vehicle type				
Route A Cars	22.82 (2.69)	15.07-26.89	22.98 (1.09)	19.99-24.80
Route A Trucks	17.85 (2.39)	11.98-21.94	18.12 (0.81)	16.29-20.03
Route F Cars	19.18 (0.54)	18.15-20.35	18.48 (0.31)	17.75-19.31
Route F Trucks	17.72 (0.37)	16.71-18.44	16.95 (0.37)	16.23-17.88
Average time-in-system per route per vehicle type				
Route A Cars	151.79 (30.13)	116.12-264.88	149.52 (13.02)	131.00-187.91
Route A Trucks	203.38 (44.05)	148.24-349.40	196.70 (17.94)	166.42-260.90
Route F Cars	91.88 (3.11)	86.02-99.21	94.21 (3.56)	86.49-107.48
Route F Trucks	101.54 (2.86)	97.02-110.64	104.57 (4.28)	95.87-114.30

Table 5.11 presents results of the Student's *t*-test to test the null hypothesis H_0: $M_{Short\_merge} = M_{MC\_adjusted\_ME3}$ for each of the eight performance measures obtained from the Adjusted Markov Chain (ME3). For the average speed KPIs, the absolute mean differences decreased compared to Markov Chain (ME2), although, the *t*-test for the Route F estimates still appeared significant.

For the average time-in-system KPI, the absolute differences for Route A decreased, but increased for Route F. This may be explained by the fact that the adjusted Markov Chain (ME3) has more vehicles generated for Route F than the second Markov Chain (ME2). Later vehicles in the Markov Chain (ME2) with smaller interarrival times (because of the geometric distribution) are more likely to contribute to the mean calculation (as the ones with larger interarrival values may not be generated due to the interarrival times larger than 7,200 seconds). The increased number of later vehicles in the adjusted Markov Chain (ME3) may also cause the introduction of higher time-in-system values for Route F, resulting in 2.33 seconds and 3.03 seconds increase in the average time-in-system for cars and trucks, respectively. However, it should be noted that the magnitude of these differences in average times are relatively small, only 3%.

Table 5.11. Student's *t*-test results for comparing mean performance measures obtained from the short merge model and the adjusted Markov Chain with time progression variable for 50 repetitions for Experiment ME3

	Short merge base model		Adjusted Markov Chain (ME3)			
	M	*SE*	*M*	*SE*	|*D*|	*t*(df), *p*
Average speed per route per vehicle type						
Route A Cars	22.82	0.38	22.98	0.15	0.16	-0.39(64.62), $p = .695$
Route A Trucks	17.85	0.34	18.12	0.11	0.27	-0.74(60.17), $p = .459$
Route F Cars	19.18	0.08	18.48	0.04	0.70	7.92(77.78), $p < .001$
Route F Trucks	17.72	0.05	16.95	0.05	0.77	10.42(98), $p < .001$
Average time-in-system per route per vehicle type						
Route A Cars	151.79	4.26	149.52	1.84	2.27	0.49(66.68), $p = .625$
Route A Trucks	203.38	6.23	196.70	2.54	6.67	0.99(64.83), $p = .325$
Route F Cars	91.88	0.44	94.21	0.50	2.33	-3.49(98), $p = .001$
Route F Trucks	101.54	0.40	104.57	0.61	3.03	-4.16(85.44), $p < .001$

The findings presented in Table 5.12 show that for the average speed KPI, the adjusted Markov Chain (ME3) resulted in smaller absolute differences and Kolmogorov-Smirnov Z values, which were no longer significant for Route A. Similarly, for Route A's time-in-system, the adjusted Markov Chain estimates improved although they were still marginally significant. However, for Route F, the K-S test indicated an increased difference between the distributions of the base model and adjusted Markov Chain (ME3), compared to the second experiment (ME2). The right shift in the cumulative distribution function and the higher maximum value in the range for Route F's time-in-system KPI may be due to an increase in the number of later vehicles from a more congested time of the adjusted Markov Chain (ME3).

Table 5.12. Two-sample Kolmogorov-Smirnov test results for comparing distributions obtained from the short merge base model and the adjusted Markov Chain with time progression variable for 50 repetitions for Experiment ME3.

	$\lvert D \rvert$	df	Kolmogorov-Smirnov Z	*p*-value
Average speed per route per vehicle type				
Route A Cars	.26	50	1.30	.068
Route A Trucks	.30	50	1.50	.022
Route F Cars	.62	50	3.10	<.001
Route F Trucks	.72	50	3.60	<.001
Average time-in-system per route per vehicle type				
Route A Cars	.28	50	1.40	.040
Route A Trucks	.28	50	1.40	.040
Route F Cars	.36	50	1.80	.003
Route F Trucks	.44	50	2.20	<.001

5.2.5 Mining towards a single vehicle model

In experiment ME2 and ME3, we assumed vehicles with unique gtuIds as unique model components, and therefore, having different behavior. Based on this assumption, we deconstructed the state-trace data and performed episode mining at component-level, resulting in 2,500 Markov Chain models, one for each vehicle. It can be argued that vehicles in the model can be treated as instances of the a single 'generic' vehicle atomic model component and the state space of this generic vehicle model is the complete set

of states observed in the training data set. In this line of reasoning, a generic vehicle model's driving behavior is imposed by the distribution of the demand (i.e., the demand for truck and passenger car), the generation times, and the surrounding traffic. In Markov Chain terms, this 'generic' vehicle component would have single 'generic' transition matrix that would contain every observed state-transition in the training set and the calculated state-transition probabilities across all 50 runs. Using this 'generic' transition matrix *P*, the complete state-space *S* and the initial distribution vector π_0, a hypothesis would be that an overarching Vehicle Markov Chain can be generated using our method and the simulation of this overarching vehicle Markov Chain would generate sufficiently accurately estimates of the base model's KPIs.

However, the traffic simulated by an overarching Markov Chain generated by our method may not be an accurate representation of the base model because the overarching model cannot guarantee consistency across the contextual variables of the generated (and currently present) vehicles. Although this issue might be addressed by introducing additional extensions to the Markov Chain implementation (e.g., including conditional probabilities and increasing the *maxwin* parameter value of the EMMA algorithm to extend state-transition history, or implementing (or mining) a control logic to oversee traffic and ensure behavioral consistency), we believe this would violate some of our fundamental assumptions, such as the memoryless property shared between Markov Chains and DEVS models. Future studies may explore alternative methods for model abstraction in situations where a longer history of events may be required to adequately represent the base model's behavior.

5.3 Conclusions

This chapter used the short-merge microscopic traffic simulation model as a case study to show how our proposed method can be applied to a simulation model with a relatively larger number of model components and a larger state space, and with a varying number of model components. We began the chapter with a high-level description of the short-merge model before delving into the specific scenario used in the case study. We then presented the conceptual model's details and described the characteristics of various

vehicle types (i.e., passenger cars and trucks) and the road network including an on-ramp to explain the effect of the merge point to the lane capacities and driving behavior.

We then addressed all considerations and actions for the modeler following the same section structure as in Chapter 3 and 4. For the larger case study presented in this chapter, we chose to apply frequent episode mining to the data represented at vehicle level in addition to mining at system level. This is because the state-trace records generated from models with large numbers of individual model components are more likely to contain high variability, limiting the identification of frequent episodes. Reducing the variability by applying frequent episode mining at component level (in this case vehicle level) instead of system level addresses this issue and allows for manageable model abstraction.

However, mining at component level omits insight into how individual components interact with and affect each other, i.e., the behavior of the system as a whole. Therefore, it is essential to include contextual variables with information on the relations among model components, such as the link or lane traffic intensity variables providing the surrounding traffic information as was done in Experiments ME2. With this information included, Markov Chains based on individual model components are able to collectively represent the overall system's behavior.

The results show that our method is able to obtain Markov Chain estimates of performance measures that do not significantly deviate from the base model's performance measures, as indicated by the *t*-test results. Furthermore, we demonstrated that the Markov Chain's ability to estimate the distribution of the base model's performance measures can be improved by adding more information to the state-trace (e.g., time progression), as shown by the Kolmogorov-Smirnov test.

However, the case study results also highlighted that a number of limitations of our method needs to be addressed by the modeler when applied to an open system simulation model with a large number of model components.

Firstly, the underlying memoryless geometric distribution used by the discrete-time Markov Chain, which determines the necessary number of state transitions for the first occurrence of a particular state does not provide a sufficient precision on the estimation of the time-dependent behavior (e.g., the arrival of a particular vehicle) of the original

system if the essential temporal information is missing in the model. The difference between the base model's Erlang distributed values and the Markov Chain's geometrically distributed and right-tailed estimates get larger when the number of entities entering the system gets significantly larger over time, leading to an extremely large sum of state-transition probabilities before the desired n^{th} state occurs. We demonstrated that the introduction of contextual variables to the data that provides the necessary temporal information (e.g., time progression) can reduce this difference. However, the inclusion of additional variables to the state-trace data may result in a larger transition probability matrix and increased variability, and may result in a longer right-tail for larger models. The trade-off between adding more information and the resulting tail should be explored.

Secondly, the case study results show that certain state-variable values with a particular significance in the base model's behavior are essential for an accurate representation of the original system's behavior and the modeler must identify and ensure that these values are explicitly represented in the selected bins during the preprocessing step. The modeler can use the model's state-trace metadata (see Appendix C.1) which corresponds to the source system in Klir's GSPS (Klir, 1985) and select from a large number of variables per vehicle or contextual variables, such as the traffic intensity variables used in ME2.

When generating state-trace data from an open system simulation, the inclusion of placeholder values such as NA to represent the inactive periods of model components will result in underfitting, especially when the run time (or the sampling period) of the simulation is significantly longer than the active period of components. As a result, the Markov Chain will have significantly smaller state-transition probabilities to transition to the actual (active) states of the model components.

Taken together, this chapter demonstrated that our method is not only capable of automating the abstraction of small models with limited components and interactions, but can also be successfully applied to larger-scale and more complex simulation models with much larger state-spaces and a varying number of model components to recreate model behavior in abstracted models and estimate the performance indicators from these abstracted models.

Chapter 6

Conclusion

6 Conclusion

With the improved capabilities of computer technology, we have been able to run simulation models that are larger in scale and higher in complexity (Davis & Bigelow, 1998; Zeigler et al., 2000). While these advances have allowed for more accurate representations of increasingly large-scale and complex real-world systems, the growing scale and complexity of simulation models may eventually result in models that become too complex themselves to work with (Astrup et al., 2008; Chwif et al., 2000; Darema, 2004; Henriksen, 2008; Saysel & Barlas, 2006) – giving rise to large-scale complex simulation models. These types of models raise important new questions and challenges for the modeling and simulation community (Arthur et al., 1999; Chwif et al., 2000; Page et al., 1999; Robinson, 2001), including how models of such large scale and complexity can be expressed, modeled more efficiently, validated, and what tools and techniques can be used for this. As described in Chapter 1, the main challenges of large-scale complex simulation models can be classified as the *problem of scale* (related to the number of objects in the model), the *problem of complexity* (related to a high resolution of objects and large number of object interactions), the *problem of performance vs. accuracy* (related to the trade-off between the level of accuracy of a simulation model and the computational cost of its execution), and the *problem of data* (related to the growth in variety and length of specifically state-trace data, limiting the ability to identify frequent behavioral patterns; Page et al., 1999, pp. 1509-1510).

A strategy that aims to tackle the scale and complexity of large-scale complex simulation models is the use of *model abstraction* (Barton, 2015; Kleijnen, 1987). As we have discussed in this dissertation (and will reflect on when answering research question 1), conventional modeling methods and techniques lack the mechanisms to efficiently and effectively deal with the aforementioned problems and are limited in their ability to automate the process of model abstraction. We therefore posit that there is a need for methods that can help reduce the complexity of large-scale and complex simulation models.

In this dissertation, we aimed to investigate to what extent the abstraction of large-scale complex simulation models, specifically discrete-event simulation models, can be automated using their state-trace data. In order to achieve this objective, we designed a novel

method that integrates the fields of modeling and simulation and temporal data mining by applying frequent episode mining techniques on state-trace data to identify behavioral patterns. In Chapter 3, we presented a detailed breakdown of our method and discussed a range of considerations that are essential for generating a valid abstraction of the base model (illustrated in Figure 6.1). We then demonstrated the practical application of our novel method using three simulation case studies with increasing scale and complexity and with different model characteristics (see Table 6.1 for a detailed breakdown of the case studies). Specifically, we investigated the ability of our method to automate the abstraction of large-scale complex discrete-event simulation models in the form of Markov Chains and to adequately estimate the original model's performance indicators. In this final chapter, we provide a discussion of our findings to answer the research questions, reflect on the theoretical and practical contributions of our study, and provide directions for future research.

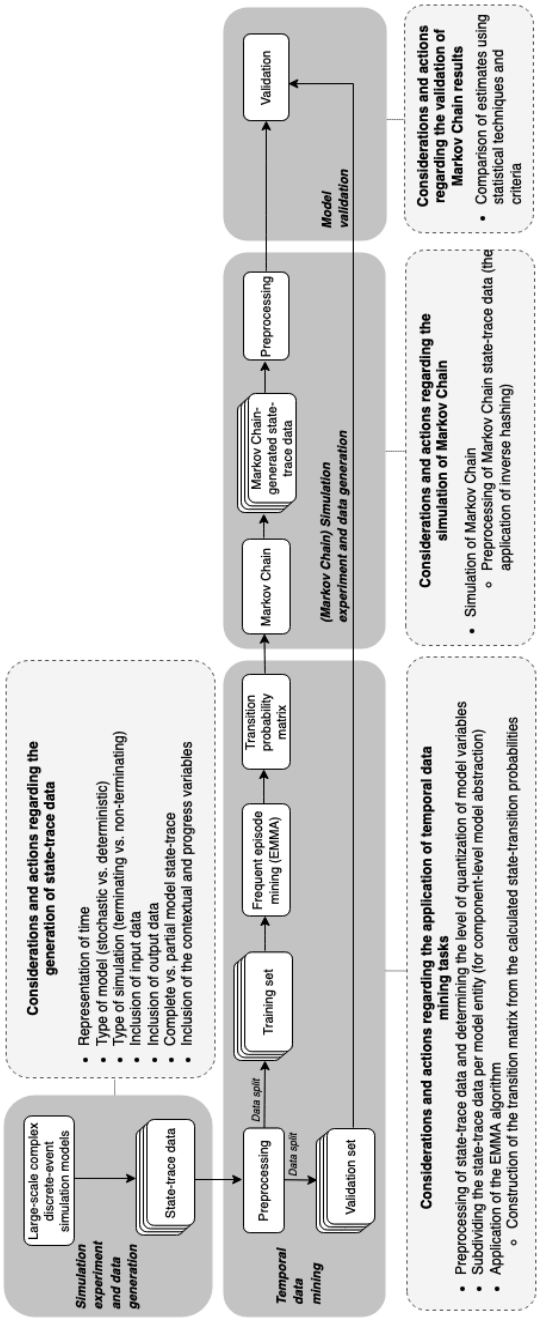

Figure 6.1. Automated model abstraction method and the organization of the considerations and actions

6.1 Research Findings

Research Question 1: *To what extent do existing methods allow for the abstraction of large-scale discrete-event simulation models?*

As briefly mentioned in the previous section and explained in more detail in Chapter 1, conventional model abstraction methods have shortcomings when dealing with large-scale complex discrete-event simulations. One of the model abstraction strategies is **metamodeling**. Providing "a model of a model" (Kleijnen, 1987), a metamodel replaces an expensive simulation model with another model that is generated by approximating the I/O function of the original one, e.g., as a set of linear equations with interaction effects. Metamodels are, therefore, simpler and computationally more efficient models than the original models (Kleijnen, 2015; Simpson et al., 2001). However, a fundamental limitation of this black box approach is that metamodels do not benefit from the time and state transition information that is present in the underlying simulation model and that describes the dynamic behavior of the system (Nance, 1981). Therefore, metamodels cannot predict I/O relations well for the set of inputs (interventions) that have not been used in estimating the metamodel's parameters.

Another model abstraction strategy is **multiresolution modeling** (MRM). In the context of large-scale complex simulation models, it is often impossible to fully capture the totality of the complex system in one abstraction (Hofmann, 2004; Yilmaz & Ören, 2004). In such cases, multiresolution modeling can be used to build a family of models (base/lumped model pairs) with different levels of abstraction or resolution that collectively represent the underlying system (Davis & Bigelow, 1998; 2003; Davis & Tolk, 2007). This approach to recursively forming model pairs allows creating a hierarchy of models with varying state trajectories, which, as a whole, provide a more complete description of a system than a single model description (Zeigler, 2019). However, until now, multiresolution modeling required modelers to design lower resolution models manually as there was no strategy to automate this process. Due to the vast and diverse number of model components and their interrelations, the manual abstraction of large-scale complex simulation models at the structure level – as most conventional model abstraction methods do – is a not a feasible task (Yilmaz & Tolk, 2006).

A more viable alternative approach for large-scale complex simulation models may be **model abstraction** at the transformation level, theoretically corresponding to the generative system level in Klir's (1985) system knowledge framework and the state transition level in Zeigler's System Specification (Zeigler et al., 2000). Abstraction at the transformation level can be achieved using state-trace data that encapsulate the model's dynamic behavior. State-trace data describe the sequential state-transitions of the model at discrete points in time, allowing modelers to capture the history of a simulation run and gain insight into how complex phenomena evolve over time (Kemper & Tepper, 2007). These data can be expressed as event sequences or multivariate time series consisting of categorical variables, numerical variables, or both. As such, state-trace data can be used to discover behavioral patterns relevant to the desired level of abstraction. Assuming a morphism relation between a pair of system specifications at the state-transition level, a correspondence relation (mapping) can be established between the base model's state-transitions (i.e., the more detailed system specification) and the lumped model's state-transitions (i.e., the simplified system specification), which uses these previously discovered behavioral patterns as aggregated states.

However, traces of state-transitions obtained from the runs of large-scale complex simulation models can get extensive in terms of their volume (the length of the trace data and the number of state variables to be sampled from different model components) and variety (the number of unique states), as we have seen in Chapter 5 where the state trace of a single simulation run of a moderately complex model had 18 million model state records; an alternative representation had 17,500 data columns. This consequently confines modelers' ability to identify and utilize frequent patterns for model abstraction. Data mining (Atluri et al., 2018; Gan et al., 2017) and machine learning methods (Pedrycz & Chen, 2014) have been designed to ease the process of discovering frequent patterns in temporal data (Hinton & Salakhutdinov, 2006). Although such methods have proven to be useful for recognizing behavioral patterns within large volumes of trace data (Chapela-Campa et al., 2019; Lu et al., 2019; Song et al., 2009; van der Aalst, 2011), they have not yet been applied to automate the abstraction of large-scale complex discrete-event simulation models. This would require techniques that not only identify important

behavioral patterns in state-trace data, but also generate aggregated states at various abstraction levels to construct models at a higher level of abstraction.

In sum, existing methods in the field of modeling and simulation use techniques that do not allow for the automated abstraction of large-scale complex discrete-event simulation models. An alternative approach to the existing methods that is explored in this thesis is to simplify the dynamic behavior encapsulated in the state-trace data of these models. To achieve this, tools and techniques from the temporal data mining field, specifically frequent episode mining, may be employed to these state-trace data to automate the model abstraction process at the transaction level. Therefore, integrating the fields of modeling and simulation and temporal data mining may provide a promising direction to deal with large-scale complex simulation models.

Research Question 2: *How should state-trace data from large-scale complex discrete-event simulations models be prepared to be used for the automated abstraction method?*

In essence, the quality of the discovered behavioral patterns by the temporal data mining techniques, and therefore, the success of our method to automate the generation of valid model abstractions is highly dependent on the state-trace data generated by the base models – the large-scale complex discrete-event simulation models. To our knowledge, no studies to date have applied temporal data mining techniques to state-trace data collected from large-scale complex discrete-event models to build valid lumped models. Because of this gap in the literature, it was essential that we first provided a formal description of the key concept of state-trace data generated by discrete-event simulation models (see §3.1.1). This helped to define our research's and our method's capabilities and properly position the method in the modeling and simulation and temporal data mining literature.

As defined in §3.1.1, a state-trace of a discrete-event simulation model is a time sequence of state-trace records (recorded instants), where each state-trace record within the same state-trace data is a fixed-size ordered set of categorical, numerical, or hybrid variables reported by the discrete event simulator at every event occurrence or at a fixed time-increment during a simulation run. To practically generate the state-trace data from the simulation runs of the case study models, we implemented a TraceWriter class that

creates state-trace data in csv format. In the data, the rows representing the state-trace records are homogenous; that is, each row has the same number of base model variable values (i.e., the values of state variables, input/output variables, and time variable).

As we discussed in Chapter 3, there are several factors that the modeler needs to consider before deciding on the content of the state-trace data and its generation from the simulation of discrete-event simulation models. In this research, we have identified and elaborated on the following important considerations:

(I) **Representation of time** (§3.2.1.1): How the simulation time can be embedded in the state-trace data is dependent on the time advance mechanism *ta* of the discrete-event simulation (base model): we can distinguish *next-event time advance (progression)* and *fixed-increment time advance*. The time advance mechanism of the model defines the limits of how frequent, or infrequent, the time-dependent behavior (i.e., time-dependent sampling of state-trace records in our method) of the original model can be captured and how the time variable can be stored in the data. Another important consideration regarding the representation of time relates to the format (elapsed time or absolute simulation time) and the variable types (categorical, numerical, hybrid) of the time information.

For example, the M/M/1 model used in the case study in Chapter 3 and the battlefield model used in Chapter 4 both have a next-event time advance mechanism. Therefore, each state-trace record is reported by the trace writer of the model at each simulation event. The numerical time variable values in the state-trace records are reported as elapsed time. The reason for the selection of the elapsed time format is twofold. First, the representation of absolute simulation time would result in monotonously increasing values in the state-trace data. Inclusion of such variables with as many different values as the number of state-trace records in the data (each value occurring only once) will undermine the ability of the frequent episode mining algorithms to find recurring patterns. Second, it represents the time advancement between episodes with a window size of 2 discovered by the EMMA algorithm. This correlates with the state transformation nature of discrete-event simulation

where δ_{int} : S→S at each event, where the time-to-next event (i.e., elapsed time) is also determined purely by the state *ta*: S→T.

The short-merge case study model in Chapter 5, however, has a fixed-increment time advance mechanism (where the increments for different vehicles are not hapening at the same instant) and the trace writer of the model generated a new state-trace record at every 0.5 seconds. On the other hand, because the step-size is constant at 0.5 seconds, the inclusion of simulation time will not affect the variability of the episodes discovered by the episode mining algorithm. Therefore, we excluded the absolute simulation time from the state-trace data in the three case studies. It should be noted that this exclusion of the absolution of time from the state-trace data led to issues, such as the inability to relate certain episodes to a time 'early' or 'late' in the simulation. To overcome such issues, we later introduced progression in the form of contextual variables.

(II) **Type of model (stochastic vs. deterministic)** (§3.2.1.2): Large-scale complex models are usually stochastic because large-scale complex systems exhibit inherent uncertainty properties. Therefore, the essence of this consideration is to identify a sufficient number of repetitions and the run-length needed to obtain independent and identically distributed observations (samples) to accurately estimate the associated variability. What constitutes a sufficient number of repetitions and run-length will depend on the model under study and the desired level of precision (i.e., the margin of error).

For the M/M/1 case study, we measured the effect of increased run-length and repetition on the precision of the performance indicator estimates by calculating the margin of error. Using a fixed state-trace length of 50,000 state-trace records, we showed that a high level of precision (i.e., below 1%) was already achieved with as few as 10 repetitions for the KPI server utilization. However, for the average waiting time and average queue length measures, the margin of error became smaller than 1% from 100 repetitions onward (note that the actual threshold of 1% is somewhere between 50 and 100 repetitions). In terms of run-length, when the number of repetitions was fixed at 100 runs, a high level of precision (i.e., below 1%) was achieved with a state-trace as short

as 1,000 records for the server utilization KPI, while for the average waiting time and average queue length measures, this level of precision was achieved from 50,000 state-trace records onward (note that the actual threshold of 1% is somewhere between 20,000 and 50,000 state-trace records). Based on these findings, we selected 50,000 as the state-trace length to be generated in conjunction with 100 repetitions. For the battlefield case study in Chapter 4, we used 50 repetitions to generate training data (and 50 repetition for generating validation data). For the chosen number of repetitions (50), we achieved margin of errors between 1.46%-4.59% across KPIs. Because the battlefield is a terminating simulation model (see also consideration (c) below), increasing the run-length is not possible. In this case, to improve the precision of the estimates, the modeler can increase the number of data samples (i.e., state-trace records) by increasing the number of repetitions with the same input set.

(III) **Type of simulation (terminating vs. non-terminating)** (§3.2.1.3): When designing a simulation experiment, in addition to deciding on the length of a simulation run (i.e., sample size), the modeler should identify the starting conditions of the model, and decide whether to include or exclude a warm-up time. These considerations are largely determined by the type of the simulation: terminating or non-terminating. For a given fixed initial state, a stochastic terminating simulation run generates state-trace data at varying lengths for each repetition during a simulation experiment. Therefore, the modeler must ensure that there are enough repetitions to compensate for (possible) shorter runs when generating state-trace data from terminating simulations. As mentioned in the previous consideration (b), for the battlefield case study in Chapter 4 we used 50 repetitions to generate training data (and 50 repetition for generating validation data). For the chosen number of repetitions, the total number of state-trace records obtained from the simulations over 50 runs were nearly identical, that is, 32,610 for the training set and 32,714 for the validation set.

When the base model is a non-terminating simulation, however, there are additional considerations. Firstly, because non-terminating simulations have no end-state or end-time and the simulation could theoretically continue

indefinitely, the modeler should decide between having fewer repetitions with longer run lengths or more repetitions with shorter run lengths. For the M/M/1 model in Chapter 3, state-trace records were generated at each event occurrence until the simulation was halted by scheduling a special "terminating" event at 1,000,000-time unit. We determined that this length is sufficient to generate at least 50,000 state-trace records for a single run. Recall from the case study application of §3.2.4 that both strategies (i.e., either keeping the run-length fixed and increasing the number of repetitions or keeping the number of repetitions fixed and increasing the run-length) provided the desired precision of less than 1% across all KPIs of the non-terminating M/M/1 model, as long as there were enough independent repetitions and sufficiently long enough runs.

Secondly, when generating state-trace data from non-terminating models, the modeler should choose one of the following strategies to reduce the effects of initialization bias: (1) to collect state-trace data from the original model for both the transient and the steady-state period and use the data to generate two Markov Chains for the transient and the steady-state period, but only perform data analysis on the steady-state period Markov Chain; or (2) not to simulate the transient period and, instead, introduce a single admissible initial state which then becomes the initial state of the steady-state period Markov Chain. Because the main subject of our method is large-scale complex discrete-event simulation models, the second option is often the more inexpensive strategy. Therefore, we implemented the second strategy for both the non-terminating M/M/1 (Chapter 3) and the short-merge case study simulations (Chapter 5). For example, for the M/M/1 case study, we introduced the initial state P_0: empty-and-idle.

(IV) **Inclusion of input data** (§3.2.1.4): The decision to include the base model's input variables as individual columns in the state-trace data and, if so, which variables to include is determined by whether the base model has external inputs and whether the simulation experiment, which consists of multiple runs to generate state-trace data, is performed with a single (fixed) set of input. When the simulation experiment consists of a single (fixed) set of input and the model

receives no external input, the exclusion of the input variables from the state-trace data does not affect the Markov Chain estimations, as demonstrated by all three case studies in this research.

(V) **Inclusion of output data** (§3.2.1.5): In our method, we presented two strategies to include the output data of the base model: (1) the direct incorporation of the transient values of the base model's run statistics as individual columns in the state-trace data, and (2) the inclusion of the state variable values of the base model, which are required to calculate the key performance indicators, directly in the state-trace data. The drawback of the first strategy is that the inclusion of the 'quantized' transient values of the base model's run statistics in the state-trace data, and therefore, a part of the Markov Chain state set, would result in a loss of sequential progression information. This can lead to inconsistencies in the representation of KPI progression in Markov Chain generated state-trace data. In the second strategy, once the Markov Chain state-traces are generated, the performance indicators are calculated by using the included model variables. We showed in all three case studies that the Markov Chain's KPIs can be sufficiently accurate as long as the state-variables are accompanied by the necessary contextual variables (see Table 6.1 for the list of variables included to calculate the KPIs for each case study).

(VI) **Complete model state-trace vs. partial model state-trace** (§3.2.1.5): A state-trace is considered as *complete-model state-trace* if it contains the complete set of variables that describe the state of all individual atomic components and, as a collection, the state of the system at a particular instant. Meanwhile, it is a *partial-model state-trace* when:

- the state-trace data have a (selected) subset of state variables of all atomic components of the base model, or
- the state-trace data have the complete set of state variables of a (selected) subset of atomic components of the base model, or

- the state-trace data have a (selected) subset of state variables of a (selected) subset of atomic components of the base model.

For the M/M/1 case study (Chapter 3), we generated a complete-model trace data because of its simplicity with only two state-variables. For the battlefield and short-merge case studies (Chapter 4 and 5, respectively), we generated partial model state-trace data (see Table 6.1 for the content of the state-trace data for all three case studies). For example, in the battlefield case study, we excluded the cell atomic components from the state-trace data and only captured each section's host cell height values, which requires only two variable values compared to 10,000 cells. Furthermore, we excluded some of the variables in the base models and only included the variables that are sufficient for the estimation of the selected performance measures (see §4.2.1.6 for more details on the selection process of these variables).

In addition to the above-mentioned considerations and actions, it is important to note that calculating key performance indicators from the Markov Chain generated state-trace data and from the base model's validation set using the included model variables alone may not always be possible. In such cases, contextual information that describes correlations between the model variables and represents a particular behavior of the base model should be captured and transformed to one or more variables in the trace data. For example, in the M/M/1 case study (Chapter 3), we initially included the variables *server_status, queue_length*, and the elapsed time in the state-trace data to calculate the key performance indicators "average waiting time," "average server utilization" and "average queue length". However, not all events in the state-trace data were events where an entity leaves the queue (e.g., arrival of a new entity). To be able to accurately calculate the average waiting time KPI, knowledge regarding leaving or non-leaving events is essential. In order to differentiate the non-leaving events from the other registered events, we therefore had to include the "leaving" behavior and information (separation of the leaving and non-leaving events) in the state-trace data. We transformed and reported this information in a new contextual variable called *time_in_queue*, which registers non-leaving events as -1 (see §3.2.1.5).

The case studies also highlighted that the inclusion of additional contextual variables reduced the margin of error in the Markov Chain's estimates and improved the accuracy of the Markov Chain's representation of the base model. For instance, in the battlefield case study (Chapter 4), we compared the margin of errors obtained from two Markov Chains MC1 and MC2. The case study demonstrated that the addition of the contextual variables *distanceToCheckpoints* and *checkpointId* to the state-traces in the training dataset of MC2 reduced the margin of errors from 2.39%-5.04% (MC1) to 1.46%-4.59% (MC2) across KPIs. Furthermore, the addition of these two contextual variables reduced the dispersion of the run lengths of the Markov Chain and resulted in estimates of the run-lengths closer to the base model. As discussed in §4.2.4, the accuracy of the Markov Chain's estimates for the base model's KPIs (i.e., total elevation climbed and total active mission time) depends on the correct sum of speed and elevation values obtained from the Markov Chain generated state-trace data, thus, the lengths of these state-traces. Because the battlefield model is a terminating simulation, reliably and accurately estimating the KPI values is therefore only possible if the Markov Chain accurately represents the terminating behavior of the base model. We showed that adding the contextual variables in MC2 helped to accurately estimate the KPIs.

Research Question 3: *What considerations and actions are needed for the automated abstraction of the original model's dynamic behavior using state-trace data?*

For the automated abstraction of the recurring patterns in the state trace data, a frequent episode mining algorithm has to be selected for the task of discovering serial state-transitions and generating the transition probability matrix P of the discrete-time Markov Chain. We compared several popular frequent episode mining algorithms and found that, at the time of our research, EMMA is the fastest in finding all unique state-trace records and consecutive state-trace record pairs from state-trace data of various lengths (5,000; 10,000; 25,000; and 50,000 state-trace records; see Appendix A.1). In all three case studies, EMMA was applied successfully.

We discussed several considerations regarding the application of the temporal data mining tasks to the state-trace data:

(I) **Preprocessing of the state-trace data** (§3.2.2.1): After the generation of state-traces from the simulation of the base model, the data must be preprocessed before the application of the EMMA algorithm. The following are the requirements for preprocessing data in our method, as well as the preprocessing methods to be used to meet these requirements:

- EMMA input format requirements: To comply with the input format of the EMMA algorithm, we created a hash table that associates each unique state-trace record (value) with an increasing positive integer value (key). This hash table will later be used to decode the hashed values from the Markov Chain generated state-traces back to their multi-variable value formats. This encoding and decoding of the state-trace records is a lossless process, and therefore, does not affect the model abstraction results.

- Quantization of high-variety data: Before the preprocessing and encoding of each state-trace record (described above), state-trace records consist of multiple model variables with categorical (binary, nominal or ordinal) or numerical values (continuous or discrete). For large-scale complex discrete-event simulations, the state-space captured as the cartesian product of the number of variables and the range of values can easily become extremely large and impractical to work with because of the low frequencies of the observations, resulting in a lack of frequent episodes. The variable types that typically lead to high variety are continuous variables, variables with monotonously increasing or decreasing values, and categorical variables with a large number of discrete categories. To avoid this issue of high variety, the data should be quantized before applying the frequent episode mining algorithm. The choice of the quantization strategy and the level of quantization depend on the level of precision required by the model abstraction objective (§3.2.2.1).

For instance, in the M/M/1 case study (Chapter 3), we chose binning as the strategy to quantize the following three variables: *elapsed_time*, *time_in_queue*, and *queue_length*. Specifically, for the *elapsed_time* and *time_in_queue* continuous variables, we used quantile-based binning based on deciles (q = 10) to create 10 bins for each variable. This is because both variables have right-skewed distributions and a strategy such as equal-width binning would result in irregularly filled bins with some bins containing a small numbers of values. Unlike the *elapsed_time* and the *time_in_queue* variables, the *queue_length* variable takes integer values where the frequencies are large for values that are close to zero; e.g., the queue length 0 has a much higher frequency than 1, and 1 is more frequent than 2, etc. As a result, using quantile-based or equal-frequency based binning would lead high frequency values to occupy more than one bin. Therefore, for the quantization of the *queue_length* variable, we chose a custom binning strategy in which we set a minimum threshold of 5% of scores per bin. This way, frequent values (>5% of scores) are assigned to their own bin, and infrequent values (<5% of scores) are grouped together until each bin contains at least 5% of the data (see §3.2.2.1).

To investigate the effect of the level of quantization to the Markov Chain's accuracy, in Chapter 4's battlefield case study, we compared lower resolution Markov Chains to higher resolution Markov Chains in which we increased the level of binning. Our findings indicate that a higher resolution alone did not improve the accuracy of the Markov Chain's representation of the base model's behavior. Only in combination with contextual variables in the training data (i.e., MC2, as discussed in research question 2), a higher resolution resulted in improved estimates.

- System state-level vs. model component-level abstraction: State-trace data generated from discrete-event simulations consist of state-trace records, reported at a particular instant in time (e.g., at event

occurrences or at fixed time intervals) during the simulation. Therefore, state-transitions of the base model are represented by consecutive state-trace record pairs in the state-trace data. This, however, changes when the state-trace data is subdivided during the preprocessing stage to accommodate for model abstraction at component-level – as illustrated by the short-merge case study in Chapter 5.

Performing model abstraction at component-level using the state-trace requires preprocessing the data to a format the EMMA algorithm can process. To do so, component-based trace data are generated by selecting the relevant columns for individual entities (e.g., vehicle) and combining them with the contextual columns that are the same across all entities (e.g., vehicles). This process creates one data set per entity (e.g., 2,500 separate data sets for the short-merge case study in Chapter 5).

(II) **Application of the EMMA algorithm** (§3.2.2.2): The end goal of the temporal data mining step of our method is to obtain the state-transition probabilities across all the state-traces in the training dataset. To compute these state-transition probabilities, we use *maxwin* values of 1 and 2, and a *minsup* value of 1 for the EMMA algorithm to discover: (a) all individual states (i.e., state-trace records) and their frequencies, and (b) all unique state-transitions (i.e., consecutive state-trace record pairs) and their frequencies. Then, using the discovered states and state-trace records and their frequencies, we can calculate the state-transition probabilities for the entire training dataset.

(III) **Construction of the transition matrix** (§3.2.2.2): After obtaining the transition probabilities, the next step is to represent these probabilities in a sparse matrix format in order to generate the discrete-time Markov Chain's transition matrix *P*. We established in Chapter 3 that the memoryless property of Markov Chain states is akin to the discrete-event model states, and the state-transition probabilities represented in matrix format is the transition matrix of a valid stochastic discrete-time Markov Chain. We showed using the case

studies that the states and the state-transitions of the base model can be mapped to valid Markov Chain states and transition matrices.

After the generation of the sparse transition matrices, the next step is to simulate the Markov Chains by performing random walks. The number of repetitions to perform with a Markov Chain to generate trace data is dependent on the number of repetitions performed by the base model to generate the validation dataset. In the M/M/1 (Chapter 3) and battlefield (Chapter 4) case studies where the modeled systems' behavior was represented by a single Markov Chain, Markov Chains were simulated 100 and 50 times, respectively. On the other hand, in the short-merge case study (Chapter 5), each Markov Chain represented the behavior of a particular vehicle as the goal was to perform model abstraction at component-level. Therefore, capturing the modeled system's behavior required simulating all 2,500 Markov Chains 50 times; equal to the number of repetitions used by the base model to generate the validation data.

When simulating each Markov Chain, the modeler should use different random number generator seed values to obtain iid data. If the base simulation is non-terminating, the modeler should terminate the data generation from the Markov Chain once a desired number of state-trace records are generated (see the "Preliminary analyses: Selection of minimum state-trace length" in §3.2.4 for more details). If the base model is a terminating model, the accumulated probability of the Markov Chain transitioning to the modeled system's end-state should be 1 after n state-changes (and this n should be close to the base model's state-trace length (see Table 4.9 in §4.2.4 for a detail analysis).

A behavior we observed in both the M/M/1 (Chapter 3) and the short-merge (Chapter 5) case studies is that some Markov Chain runs terminated prematurely, before the desired number of state-trace records were generated (i.e., 50,000 or 7,200). We discovered that this is because some of the single system-level Markov Chain models of the M/M/1 simulation and vehicle-based Markov Chains of the short-merge simulation are not complete graphs because of the data collection strategy. As mentioned earlier in Chapter 3 and discussed in the answer of the research question 2, the non-terminating simulations have to be artificially terminated to stop the data generation. As a result, some of the state-trace data generated from the runs inherited an artificial terminating state (could be a different state for each of these runs) because the final state did not occur earlier in the

run. Consequently, some of the transitions in these right-stochastic transition matrices will have the probability of 1, meaning that the previous states will always lead to these artificial end states and the run will terminate. In some cases, the transition matrix might have a chain of states transitioning to each other with the transition probability of 1. There are multiple ways to deal with artificial terminating states in the transition matrix. The first approach is to identify and delete all artificial terminating states until the graph becomes a complete graph and recalculate the state-transition probabilities after the removal of every artificial terminating state. However, this approach may result in the removal of a large number of states from the transition matrix (due to the chaining as we just mentioned). The second approach is to use a portion of the generated state-trace data in the episode mining algorithm to generate the transition probability matrix. For instance, the modeler may decide to use 80% of the total state-trace data for the training and validation set. In this case, the modeler must ensure that the remaining data contain enough samples to generate valid Markov Chains with a desired level of precision. To avoid having an insufficient number of samples, the modeler can first perform longer runs to generate more data and subsequently use a portion of the data generated from these longer runs.

Research Question 4: *How well does the abstracted model represent the behavior of the original large-scale complex simulation model?*

To evaluate the accuracy and the validity of the abstracted model, key statistics and performance measures obtained from the original model's validation set and the abstracted model (Markov Chain) can be compared using descriptive (e.g., spread measures, histograms, box and whisker plots) and inferential statistics (e.g., χ^2-test, Student's t-test, two-sample Kolmogorov-Smirnov test).

The results of the case studies, and specifically their validation process, demonstrated that our method is able to obtain Markov Chain estimates of performance measures with an acceptable level of precision that do not significantly deviate from and follow a similar distribution as the original model's validation set. In Chapter 3, with a relatively simpler M/M/1 case study, we found that our Markov Chain's estimates had a high level of precision and were highly similar to the validation set's estimates. In Chapter 4, with the battlefield case study, we demonstrated that our method can generate valid abstractions

of relatively larger and more complex discrete-event simulation models and can obtain Markov Chain estimates of performance measures that do not significantly deviate from the base model's performance measures. Similarly, the findings from the short-merge case study in Chapter 5 illustrated that our method is applicable to larger and more complex short-merge discrete-event simulation models and can obtain Markov Chain estimates of performance measures that are highly similar to the base model's performance measures.

As we addressed earlier in our discussion of research question 2, the presence of contextual variables is crucial for the accurate representation of the base model's behavior, and for obtaining run-lengths that are sufficiently close to the base model's run-length. In fact, our case studies revealed that increasing the model's resolution by increasing the quantization levels of the variables (e.g., the number of bins) alone did not improve the accuracy of the model in the absence of important contextual variables. Our findings in section §4.2.4 showed that a higher resolution only improved the results of the model that included contextual variables. A similar improvement in the Markov Chain's accuracy was also observed in the case study in Chapter 5.

Although we employed inferential statistics for the validation of the estimates obtained from the Markov Chains, it is of great importance for the modeler to not solely rely on p-values to judge the accuracy of the base model's representation (Lin et al., 2013) but also evaluate descriptive statistics as well as the effect size which describes the magnitude of the difference (Sullivan & Feinn, 2012). Moreover, it is important that the Markov Chain's estimates are considered within the context of the system of interest and the goals of the model abstraction study. For instance, in the short-merge case study (Chapter 5), the Student's t-test results indicated that mean speeds for the car and truck generated from the Markov Chain differed significantly from the mean speeds obtained from the base model for Route F; however, the absolute mean difference was minimal. Similarly, the results of the χ^2-test in Chapter 3 illustrated that a large sample size may result in flagging a significant difference between the two distributions (a known problem with significance testing based on p-values; Sullivan & Feinn, 2012), even though the Cramer's V effect size and graphical visualizations suggested that the magnitude of this difference was very small.

Table 6.1. Case study comparisons

	CH3: M/M/1 case study	CH4: Battlefield case study	CH5: Short-merge case study
	System properties		
System description (§3.1.4) (§4.1.2) (§5.1)	Single-queue single-server queueing system; interarrival times of entities and service times are random; entities enter the queue unless the queue is empty, seized by the server, delayed during the time of service, and leave the system when released by the server.	Battlefield system consists of a terrain with hills and passages between hills, and a tank platoon consists of two sections. The sections start their mission at time t=0 seconds and pass through three checkpoints. The mission ends when both sections arrive to the end station.	A road network system consists of a two-lane main road (Route A), a single-lane on-ramp (Route F) merging onto the main road. Vehicles (passenger cars and trucks) can enter and leave the system at any time through the system boundaries.
System type (§3.1.4) (§4.1.2) (§5.1)	Open discrete-event system.	Closed discrete-event system.	Open discrete-event system.
	Modeling Details		
Modeling formalism (§ 3.1.4) (§ 4.1.3) (§ 5.1).	Event-scheduling world view	Coupled DEVS (modeled using the Dynamic Structure DEVS	Event-scheduling world view (microscopic traffic flow model)
Time advance mechanism (§ 3.2.1.1) (§ 4.2.1.1) (§ 5.2.1.1)	Next-event time advance	Next-event time advance	Fixed-increment time advance (τ= 0.5 seconds), but not synchronized across components
Stochastic / Deterministic model (§ 3.2.1.2) (§ 4.2.1.2) (§ 5.2.1.2)	Stochastic model; the case study investigated the effects of increasing number of repetitions with six different number of runs to generate iid data (Table 3.12): 10, 20, 50, 100, 200, 500; unique Mersenne Twister seed used for each repetition.	Stochastic model; total of 100 repetitions to generate iid data; unique Mersenne Twister seed used for each repetition.	Stochastic model; total of 100 repetitions to generate iid data; unique Mersenne Twister seed used for each repetition.

Terminating /Non-terminating model (§ 3.2.1.3) (§ 4.2.1.3) (§ 5.2.1.3)	Non-terminating; no warm-up period; two sets of experiments with six experiments in each set. For the first experiment set, the simulation run is terminated once the desired number state-trace records are reported.	Terminating; the simulation ends when both sections arrive to their corresponding end stations; state-trace data with varying length.	Non-terminating; a method call is scheduled to be executed before the simulation clock reaches to the predetermined end time of the simulation (i.e., 3,600 seconds).
Input data (§ 3.2.1.4) (§ 4.2.1.4) (§ 5.2.1.4)	Average arrival rate λ of 0.1 and average rate of service μ of 0.118 (Table 3.1). Single set of input with no external input.	Tank platoon section trajectories and terrain details (Table 4.5). Single set of input with no external input.	Vehicle trajectories and lane demands over a time-space region. Single set of input with no external input (Table 5.2).
Output data & Key performance indicators (§ 3.2.1.5) (§ 4.2.1.5) (§ 5.2.1.5, 5.2.1.5.1, 5.2.1.5.2)	As listed in Table 3.2: (a) Average time in system (b) Average waiting time in queue (c) Average number (of entity) in queue	As listed in Table 4.6: (a) Active mission time Section A (b) Active mission time Section B (c) Total elevation climbed Section A (d) Total elevation climbed Section B	A total of 8 key performance indicators as listed in Table 5.3: (a) Average time-in-system per route (A or F) per vehicle type (car or truck) (b) Average speed per route (A or F) per vehicle type (car or truck)
State-trace Properties			
Complete /Partial model state-trace generation (§ 3.2.1.6) (§ 4.2.1.6) (§ 5.2.1.6)	Complete model state-trace data	Partial model state-trace data	Partial model state-trace data
State variables included in the trace data (§ 3.2.1.5) (§ 4.2.1.5) (§ 5.2.1.5)	The state-trace data used in M/M/1 case study contains the list of variables state variables such as *server_status* and *queue_length* (Figure 3.5).	Two experiment sets designed with two different datasets for the case study (for MC1 and MC2). Both of these datasets contain the *speed* variable from the Section atomic model and *elevation*	Two different datasets are used for the short-merge case study. These datasets contain the state variables listed in Table 5.4.

		variable from the Cell atomic model (Table 4.7).	
Contextual variables included (§ 3.2.1.5) (§ 4.2.1.6) (§ 5.2.1.6.1, § 5.2.4.2)	The contextual variables in M/M/1 model are *time_in_queue* and *num_of_observations*, which are essential for the calculation of the key performance measures listed in Table 3.5.	In the first dataset (for MC1), we included the contextual variable called *movementIndicator* for both sections, indicating whether a reported event is a movement event or not for the corresponding section. In the second dataset (for MC2), we also included two additional variables *distanceToCheckpoint* and *checkPointID* for each section.	The datasets for Experiment ME2 contain the additional traffic intensity contextual variables listed in Table 5.6. The dataset for Experiment ME3 includes the traffic intensity variables that are also in the first set as well as additional progress indicator variables for each vehicle.
Average trace length (§ 3.2.4) (§ 4.2.1.2) (§ 5.2.1.3)	Fixed; warm-up period is represented in the state-trace data with an initial state; the case study investigated the effects of state-trace length using traces with six different lengths (Table 3.9).	Not-fixed; the length of a state-trace generated from a single run is dependent on the terminating condition (Table 4.4). The average state-trace length in the training and validation datasets (50 repetitions) are 652.20 and 654.28 state-trace records, respectively.	Fixed; 7,200 state-trace records (rows)
Pre-processing & Frequent Episode Mining Step Details			
Binning strategy (§ 3.2.2.1) (§ 4.2.2.1) (§ 5.2.2.1)	Mixed binning strategy: (a) quantile-based binning (decile) is used for the *elapsed_time* and *time_in_queue* variables, and (b) custom binning strategy with a minimum threshold of 5% of scores per bin for the *queue_length* variable.	Fixed-width binning	Fixed-width binning

Frequent episode mining algorithm / Algorithm parameters / Mining strategy	EMMA Algorithm: Minsup: 1 Maxwin: 2 Mining at system state level; each state-trace record (i.e., each row) containing the full set of system states is encoded as a single positive integer value.	EMMA Algorithm: Minsup: 1 Maxwin: 2 Mining at system state level; each state-trace record (i.e., each row) containing state variables of both tank platoon sections is encoded as a single positive integer value.	EMMA Algorithm with the addition of the skip-length variable: Minsup: 1 Maxwin: 2 Skip-length: 2,500 Mining at individual-component state level; each state-trace record containing 2,500 vehicle states is subdivided into 2,500 data sets, converting the state-trace data with 7,200 rows to 18,000,000 rows. The EMMA algorithm, then, applied to the subdivided data using a skip-length of 2,500.
Transition probability matrix details	Single transition probability matrix containing the state-transition probabilities for all *minsup=1* system state.	Two transition probability matrices generated/used for the experiments MC1 and MC2.	For experiments ME2 and ME3, 2,500 transition probability matrices generated.

In sum, the presented research showed that our novel method is capable of automating the abstraction of large-scale and complex discrete-event simulation models with large state-spaces. The findings throughout this dissertation demonstrated that the method is able to generate valid Markov Chains from the state-trace data of the base model and the calculated output performance indicators from these Markov Chains have sufficient precision and accuracy.

6.2 Main Contributions

In this dissertation, we presented a novel frequent episode mining-based method for the automated abstraction of discrete-event simulation models using state-trace data. By presenting how concepts from the modeling and simulation and temporal data mining fields

can be integrated, we contributed to an advanced understanding of how large scale and complex simulation models can be abstracted automatically, and what tools and techniques can be used in this process. Importantly, our research introduced a formal description of discrete-event simulation generated state-trace data and state-trace records which help to properly position our method in the modeling and simulation literature. Moreover, by presenting the detailed steps of our method and describing a range of considerations and actions that modelers may face through diverse case studies, we demonstrated how state-trace data and temporal data mining techniques may best be used for models with different characteristics. The findings in this dissertation are not only of scientific interest; they may also be valuable to modeling and simulation experts and decision makers aiming to understand and capture the underlying behavior of large-scale complex systems or data analysts working with big data that has similar characteristics as state-trace data.

6.3 Directions for Future Research

The findings reported in this dissertation should be interpreted in light of several limitations. Firstly, the focus of this research, in terms of the models of interest, was limited to discrete-event simulations models. Other types of models may be included in future research to test the applicability of our method to a broader range of simulation models. Agent-based models, for example, may be an interesting candidate because the time advance in most agent-based models uses equidistant intervals. The discrete-event short-merge model (Chapter 5) uses a similar time advance mechanism, in which the states of individual vehicles were recalculated at (equidistant) 0.5-second intervals. Future work may also reveal new considerations and actions for generating state-trace data from agent-based models, which could help identifying behavioral patterns of agents from the state-trace data.

Secondly, we chose Markov Chains to represent the abstracted models in our method because its memoryless property is similar to that of discrete-event model states. Moreover, the state-transition probabilities of the base model represented in matrix format correspond to the transition matrix of a discrete-time Markov Chain. However, Markov Chains may have shortcomings due to the effect of repetitive drawing of probabilities,

where the average time to reach a certain state might be correct, but the standard deviations differ very much from the underlying data. From the perspective of automated model abstraction, generation of abstract models from state-trace data using richer modeling formalisms such as port-based DEVS or Petri nets can be an interesting future research direction.

Thirdly, for this research we compared the performance of some of the popular frequent episode mining techniques and concluded that the EMMA algorithm was best suited for our method. However, the field of data mining and machine learning is rapidly expanding and new methods, tools, and techniques are introduced in a fast pace. It is therefore important that future modelers working with our method reevaluate the available algorithms and select the one most suitable for their objectives.

Finally, this dissertation demonstrated the application of our method in three discrete-event simulation case studies. Future research can explore our method's usability to other industries and actual, real-life problems, such as, global supply chains, modern manufacturing systems, and power grids.

Appendices

Appendix A

A.1. Comparison of Frequent Episode Mining Algorithms

This chapter provides further details regarding the performance comparison of three popular frequent episode mining algorithms: MINEPI (Mannila et al., 1997), MINEPI+ (Huang & Chang, 2008), and EMMA (Huang & Chang, 2008). For this comparison study, we employed the SPMF open-source data mining library's Java implementations of the three aforementioned frequent episode mining algorithms: MINEPI, MINEPI+, and EMMA (Fournier-Viger et al., 2014; 2016). The SPMF library version was 2.41, which was the most recent version at the time the experiments were carried out. We devised four experiments in which we compared the *minimum*, *maximum*, and *average execution times* of each algorithm and the *total candidate generated* from their execution against each other and for increasing file lengths (i.e., the number of rows): 5,000; 10,000; 25,000; 50,000. The algorithms are given the same input parameters of *minsup* of 1 and *maxwin* of 2 to discover all unique serial episodes with length 1 and length 2 that appears at least one time in the input data. These input parameters are also the parameters used for the episode mining task of our proposed automated model abstraction method and the discovered episodes corresponds to the unique state-trace records and state-trace record pairs – i.e., state-transitions – in the state-trace data. For the first three experiments where the input file lengths are 5,000; 10,000; 25,000, we obtained the results for each algorithm by running them 50 times. However, for the fourth experiment, where we used an input file with a length of 50,000, we limited the total number of runs to 20 for practical reasons (i.e., the average runtime of the MINEPI algorithm to generate results for the given input data set was close to one hour).

The results of all four experiments are presented in the below four tables: Table A.1.1, A.1.2, A.1.3, and A.1.4. In addition to the comparison experiment results, Table A.1.5 provides an overview of the hardware specifications of the system used in the experiments. Experiments revealed that the EMMA algorithm is consistently the fastest at finding all unique state-trace records and consecutive state-trace record pairs from state-trace data of varying length: 5,000; 10,000; 25,000 and 50,000 state-trace records. The differences in the total number of candidates generated are due to the different methods used

by each algorithm to count episode frequencies as described in (Huang & Chang, 2008). While the total number of generated candidate episodes varies by algorithm, the number of frequent episodes identified by each algorithm is the same. For the first experiment shown in Table A.1.1, the number of frequent episodes identified by each algorithm is 5,061.

Table A.1.1. Comparison of total mining times and total candidate generated for state-trace length 5,000 across 50 runs

	Minimum execution time (ms)	Maximum execution time (ms)	Average execution time (ms)	Total candidate generated
MINEPI	97,686	103,672	99,722	1,158,273
MINEPI+	341	377	355	8,867,867
EMMA	215	235	219	5,431,526

Table A.1.2. Comparison of total mining times and total candidate generated for state-trace length 10,000

	Minimum execution time (ms)	Maximum execution time (ms)	Average execution time (ms)	Total candidate generated
MINEPI	271,888	302,065	282,520	1,606,119
MINEPI+	1,023	1,084	1,050	17,261,845
EMMA	514	538	521	10,508,160

Table A.1.3. Comparison of total mining times and total candidate generated for state-trace length 25,000

	Minimum execution time (ms)	Maximum execution time (ms)	Average execution time (ms)	Total candidate generated
MINEPI	1,045,615	1,439,626	1,212,278	2,043,308
MINEPI+	4,451	5,014	4,544	35,054,252
EMMA	1,854	1,905	1,879	21,151,472

Table A.1.4. Comparison of total mining times and total candidate generated for state-trace length 50,000

	Minimum execution time (ms)	Maximum execution time (ms)	Average execution time (ms)	Total candidate generated
MINEPI	2,773,234	3,474,219	3,229,606	2,348,132
MINEPI+	14,619	15,761	14,781	53,749,181
EMMA	4,651	4,806	4,730	32,260,032

Table A.1.5. An overview of the hardware specification of the system used in the experiments

Type	Specifications
Processor Name	Intel Core i5
Processor Speed	3.1 GHz
Number of Processors	1
Total Number of Cores	2
L2 Cache (per Core)	256 KB
L3 Cache	4 MB
RAM Type	LPDDR3
RAM Size	2x4 GB
RAM Speed	2,133 MHz

A.2. EMMA Algorithm Input Format

The SPMF implementation (Fournier-Viger et al., 2014; 2016) of the EMMA algorithm has five input parameters:

(I) An input text file, which contains preprocessed state-trace data

(II) The output file, which will eventually contain identified episodes and their frequency counts

(III) Minimum support threshold (or minsup)

(IV) Maximum time duration or maximum window length

(V) A Boolean parameter indicating that the input data is ordered and each row contains a sequence number incrementing by 1 when it is set to "false".

For a given input data set, the output of the SPMF implementation of the EMMA algorithm consists of two pieces of information: (a) the set of frequent episodes with support no less than the given minsup parameter and with length smaller than or equal to the maxwin parameter, and (b) the frequencies of these episodes. To sequentially process all state-traces in the training data set, we introduced two user-defined parameters to the EMMA implementation: (a) *rangeFrom* (initialization parameter) and (b) *rangeTo* (condition parameter).

Appendix B

This chapter provides details regarding the JAVA implementation of the *MarkovModel* Class used by the case studies in Chapter 4, and 5 and the MATLAB implementation of the *MarkovChain* used by the M/M/1 case study in Chapter 3[29].

B.1. Markov Chain: Java Implementation Details

After obtaining the transition probabilities, the next step is to represent these probabilities in a sparse matrix format in order to generate the discrete-time Markov Chain's transition matrix *P*. Therefore, as shown in line 1, one of the input parameters for the MarkovModel constructor given in Table B.1 is a sparseMatrix of the particular Markov Chain. The sparseMatrix generated by the end of the frequent episode mining phase of our automated model abstraction method contains three columns (line 20-22 in Table B.1 reads these three columns): *state i*, *state j*, *transitionProbabilities Pij*. An example sparseMatrix containing some example data belonging to is given in Table B.2.

Table B.1. MarkovModel constructor

```
1  public MarkovModel(String directory, String sparseMatrixFilename)
2    {
3       try (BufferedReader sparseMatrixFile = new BufferedReader(new FileReader(direc-
4  tory + sparseMatrixFilename)))
5       {
6          String line;
7          String[] parts;
8          int s = 0;
9          int t = 0;
10         double p = 0.0;
11         int source = -1;
12         List<Double> probList = new ArrayList<>();
13         List<Integer> targetList = new ArrayList<>();
14         do
```

[29] It should be noted that source code and data will become available as open data, and that the final thesis will contain the link to all data used in the research.

```
15          {
16              line = sparseMatrixFile.readLine();
17              if (line != null)
18              {
19                  parts = line.split("\t");
20                  s = Integer.parseInt(parts[0]);
21                  t = Integer.parseInt(parts[1]);
22                  p = Double.parseDouble(parts[2]);
23              }
24              if (line == null || s != source)
25              {
26                  if (source != -1 || line == null)
27                  {
28                      int[] targetArray = new int[targetList.size()];
29                      double[] probArray = new double[probList.size()];
30                      double cumProb = 0.0;
31                      for (int i = 0; i < probArray.length; i++)
32                      {
33                          targetArray[i] = targetList.get(i);
34                          cumProb += probList.get(i);
35                          probArray[i] = cumProb;
36                      }
37                      probArray[probArray.length - 1] = 1.0;
38                      this.targetMap.put(source, targetArray);
39                      this.probMap.put(source, probArray);
40                      probList.clear();
41                      targetList.clear();
42                  }
43                  source = s;
44              }
45              targetList.add(t);
46              probList.add(p);
47          }
48          while (line != null);
49          sparseMatrixFile.close();
50
51      }
52      catch (IOException e)
53      {
54          e.printStackTrace();
55      }
56  }
```

Table B.2. Example sparseMatrix with states and transition probabilities

State *i*	State *j*	Transition Probability *Pij*
1	2	0.680
1	4	0.020
1	4683910	0.020
1	6186116	0.020
1	1108859	0.020
1	2563306	0.040
1	261035	0.040
1	261033	0.080
2	1108859	0.010
2	261033	0.069
2	261035	0.079
2	2	0.663
2	4	0.020
2	579758	0.040
2	4683910	0.020
2	861729	0.079
2	8125670	0.010
2	6186116	0.010
4	4	0.727
4	7	0.273
7	7	0.769
7	12	0.077
7	4026157	0.154
12	16	1.000
16	16	0.875
16	23	0.125
23	23	0.882

After the Markov Chain is constructed using the transition matrix, the next step is to simulate the Markov Chain. The Table B.3 shows the code snippet of the *run* method that performs the (stochastic) state walks. The method takes a unique *Mersenne Twister* pseudo-RNG seed value, a repetition count and the initial state of the Markov Chain.

Table B.3. run method of the Markov Chain that executes the state walks

```
1  public List<Integer> run (final long seed, final long maxCount, int initialState)
2    {
3       List<Integer> outputStateList = new ArrayList<>();
4       StreamInterface stream = new MersenneTwister(seed);
5       int state = initialState;
6       int count = 0;
7       double[] probs;
8       double rand;
9       int nr;
10      while (true)
11      {
12         outputStateList.add(state);
13         probs = this.probMap.get(state);
14
15         if (probs == null)
16         {
17          int lastIndex = outputStateList.size()-1;
18          outputStateList.remove(lastIndex);
19          break;
20         }
21
22         rand = stream.nextDouble();
23         if (probs[0] == 0 || count >= maxCount)
24            break;
25
26         nr = 0;
27         while (probs[nr] < rand)
28            nr++;
29         state = this.targetMap.get(Army)[nr];
30
31         count++;
32      }
33      return outputStateList;
34   }
```

B.2. Markov Chain: MATLAB Implementation Details

The creation and the simulation of the discrete-time Markov Chain (dtmc) used by the M/M/1 case study in chapter 3 is implemented in MATLAB. We used the *dtmc* object that comes with the Econometrics Toolbox[30] of MATLAB (version R2018b) to create the discrete-time, finite-state, time-homogeneous Markov chain from given state transition matrix. To simulate the random state walks of a generated *dtmc*, we used the *simulate* function which comes with the same Econometrics Toolbox. The simulate function returns state-trace data for each run in the form of random state walks for the following input parameters: 1) the discrete-time Markov Chain *mc*, 2) random state-walk length *numSteps,* 3) the initial state `'X0'` of 4) simulation `x0`. The Table B.4 shows the code snippet of the *MarkovChain.m* function which was the main program to perform Markov Chain experiments for the M/M/1 case study.

The MarkovChain function takes the transition matrix, initial pseudo-RNG seed value (e.g., 5001 for the M/M/1 case study), number of repetitions (e.g., 100 runs), a list of stateNames (i.e., unique episodes with *maxwin* of 1 identified by the EMMA algorithm), and the number of steps (i.e., the length of the state walk – 50,000 for the M/M/1 case study). It can be seen from the code snippet that the *mc* discrete-time Markov Chain is created in line 10 and simulated using the *simulate* function in line 17. Finally, the output of the *simulate* function, which is a Markov Chain state-trace, is written to text file after each run (line 26).

[30] The Econometrics Toolbox of MATLAB: https://www.mathworks.com/products/econometrics.html

Table B.4. MarkovChain.m MATLAB implementation

```
function MarkovChain(transitionMatrix, initialSeed, numOfRep, stateNames, num-
Steps)
 numStepIndex = 1;
 for seed = initialSeed:(initialSeed + (numOfRep-1))

  rng(seed ,'twister'); % For reproducibility
  %assigning TM to P. Note that TM should be loaded to the memory.
  P = transitionMatrix;

  %creation of the mc discrete-time Markov Chain
  mc = dtmc(P, 'StateNames', stateNames);

  %setting the number of states vector with [1 0 0 0 ... 0]
  x0 = zeros(1, mc.NumStates);
  x0(1,1) = 1;

  %simulating the mc discrete-time markov chain
  X = simulate(mc,numSteps(1,numStepIndex),'X0',x0);

  X = X(X~=0); %only non-zero states (sparse matrix)
  L = length(X); %Length of each run
  %Markov Chain state-trace file name
  X_MC_name = strcat('MC_StateTrace_', num2str(numStepIndex,'%03.f'),'.txt');

 %writing the content of the state-trace to the file
  fid = fopen(X_MC_name, 'wt');
  dlmwrite(X_MC_name, X,'precision', 6);
  fclose(fid);

  numStepIndex = numStepIndex + 1;
 end %end of for loop
end %end function MarkovChain.m
```

Appendix C

C.1. Short-merge Case Study State-Trace Metadata

This chapter provides further details on the *reportState* method implementation in Java that was used in Chapter 5's short-merge case study model to capture the state-trace metadata, and the content of this state-trace metadata with the model reported model variables, example values, and their descriptions. A state-trace metadata corresponds to the source system in Klir's GSPS (Klir, 1985) and contains the time-indexed trajectories of model variables such as state-variables, run-statistics, contextual variables, and simulation time needed for the possible set of model abstraction experiments to be performed. The modeler can use the proposed method to select a subset of the state-trace metadata and perform the automated model abstraction.

To iteratively identify and include new model variables such as state-variables, run-statistics, contextual variables, and simulation time to the short-merge model's state-trace data for each new model abstraction experiment (e.g., addition of new variables to improve model accuracy) is not an optimal strategy given the time it takes for a single run to generate state-trace data from the short-merge model and the desired number of repetitions (100). Instead, we generated a state-trace metadata set that includes a larger set of model variables that may be necessary for the possible set of model abstraction experiments. Then, we selected a subset of the columns from this meta state-trace data in order to perform the case study in Chapter 5. Table C.1 below lists this larger set of model variables[31] generated from the execution of the short-merge model. The table shows the name of the variables, example values, and their descriptions.

[31] The vehicle-related variables (i.e., twenty variables per vehicle) are listed only for 2 vehicles as they repeat for all 2,500 vehicles.

Table C.1. State-trace metadata of the short-merge case study

Model variable		Value	Description
1.	time	3,599.5	Absolute time
2.	elapsedTime	0.5	Delta time
3.	nrVehiclesAB_d1	30	Start glue option 1
4.	nrVehiclesAB_d2	67	|
5.	nrVehiclesBC_d1	1	| This contextual variable set indicates the
6.	nrVehiclesBC_d2	3	|number of cars on each lane. Through this
7.	nrVehiclesBC_d3	0	|value, average speed, acceleration and
8.	nrVehiclesCD_d1	1	| deceleration, as well as lane changes
9.	nrVehiclesCD_d2	0	| are influenced. The more cars, the lower
10.	nrVehiclesDE_d1	21	| the average speed, for instance.
11.	nrVehiclesDE_d2	27	|
12.	nrVehiclesF2B_d1	1	End contextual variable set - option 1
13.	nrVehiclesAB	97	Start contextual variable set - option 2
14.	nrVehiclesBC	4	| Shorter version with #cars per link
15.	nrVehiclesCD	1	| (all lanes combined)
16.	nrVehiclesDE	48	| So #13 = #3 + #4 (97 = 30 + 67)
17.	nrVehiclesF2B	1	End contextual variable set - option 2
18.	avgSpeedACar_N	1627	Start statistic #1
19.	avgSpeedACar_Mean	24.53	| Speed statistic of cars that left the
20.	avgSpeedACar_StDev	6.54	| model, for cars that started at the
21.	avgSpeedACar_Min	7.58	| left side of the model.
22.	avgSpeedACar_Max	37.84	End statistic #1
23.	avgSpeedFCar_N	312	Start statistic #2
24.	avgSpeedFCar_Mean	19.52	| Speed statistic of cars that left the
25.	avgSpeedFCar_StDev	2.513	| model, for cars that started at the
26.	avgSpeedFCar_Min	14.28	| bottom side of the model.
27.	avgSpeedFCar_Max	27.39	End statistic #2
28.	avgSpeedATruck_N	266	Start statistic #3
29.	avgSpeedATruck_Mean	18.51	| Speed statistic of trucks that left the
30.	avgSpeedATruck_StDev	4.79	| model, for trucks that started at the
31.	avgSpeedATruck_Min	7.68	| left side of the model.
32.	avgSpeedATruck_Max	25.89	End statistic #3
33.	avgSpeedFTruck_N	141	Start statistic #4
34.	avgSpeedFTruck_Mean	17.03	| Speed statistic of trucks that left the
35.	avgSpeedFTruck_StDev	1.67	| model, for trucks that started at the
36.	avgSpeedFTruck_Min	13.64	| bottom side of the model.
37.	avgSpeedFTruck_Max	20.73	End statistic #4
38.	modelTimeACar_N	1627	Start statistic #5
39.	modelTimeACar_Mean	133.64	| Time-in-system of cars that left the
40.	modelTimeACar_StDev	55.91	| model, for cars that started at the
41.	modelTimeACar_Min	77.73	| left side of the model.
42.	modelTimeACar_Max	387.68	End statistic #5
43.	modelTimeFCar_N	312	Start statistic #6
44.	modelTimeFCar_Mean	87.74	| Time-in-system of cars that left the
45.	modelTimeFCar_StDev	10.73	| model, for cars that started at the
46.	modelTimeFCar_Min	61.56	| bottom side of the model.
47.	modelTimeFCar_Max	118.10	End statistic #6
48.	modelTimeATruck_N	266	Start statistic #7
49.	modelTimeATruck_Mean	173.75	| Time-in-system of trucks that left the
50.	modelTimeATruck_StDev	61.91	| model, for trucks that started at the
51.	modelTimeATruck_Min	113.31	| left side of the model.
52.	modelTimeATruck_Max	381.53	End statistic #7
53.	modelTimeFTruck_N	141	Start statistic #8
54.	modelTimeFTruck_Mean	99.47	| Time-in-system of trucks that left the
55.	modelTimeFTruck_StDev	9.79	| model, for trucks that started at the
56.	modelTimeFTruck_Min	80.93	| bottom side of the model.
57.	modelTimeFTruck_Max	122.99	End statistic #8
58.	nrVehiclesABForward1_nrVehicles	30.0	Start statistic #9
59.	nrVehiclesABForward1_N	7199	| Number of vehicles on AB lane 1
60.	nrVehiclesABForward1_WeightedMean	16.74	| nr = number at the moment,
61.	nrVehiclesABForward1_WeightedStDev	8.28	| N = number of observations,
62.	nrVehiclesABForward1_Min	0.0	| mean, stdev, min, max are weighted stats
63.	nrVehiclesABForward1_Max	43.0	End statistic #9

64.	nrVehiclesABForward2_nrVehicles	67.0	Start statistic #10
65.	nrVehiclesABForward2_N	7199	| Number of vehicles on AB lane 2
66.	nrVehiclesABForward2_WeightedMean	32.35	| nr = number at the moment,
67.	nrVehiclesABForward2_WeightedStDev	20.40	| N = number of observations,
68.	nrVehiclesABForward2_Min	0.0	| mean, stdev, min, max are weighted stats
69.	nrVehiclesABForward2_Max	87.0	End statistic #10
70.	nrVehiclesBCForward1_nrVehicles	1.0	Start statistic #11
71.	nrVehiclesBCForward1_N	7199	| Number of vehicles on BC lane 1
72.	nrVehiclesBCForward1_WeightedMean	1.35	| nr = number at the moment,
73.	nrVehiclesBCForward1_WeightedStDev	0.95	| N = number of observations,
74.	nrVehiclesBCForward1_Min	0.0	| mean, stdev, min, max are weighted stats
75.	nrVehiclesBCForward1_Max	4.0	End statistic #11
76.	nrVehiclesBCForward2_nrVehicles	3.0	Start statistic #12
77.	nrVehiclesBCForward2_N	7199	| Number of vehicles on BC lane 2
78.	nrVehiclesBCForward2_WeightedMean	2.04	| nr = number at the moment,
79.	nrVehiclesBCForward2_WeightedStDev	1.10	| N = number of observations,
80.	nrVehiclesBCForward2_Min	0.0	| mean, stdev, min, max are weighted stats
81.	nrVehiclesBCForward2_Max	5.0	End statistic #12
82.	nrVehiclesBCForward3_nrVehicles	0.0	Start statistic #13
83.	nrVehiclesBCForward3_N	7199	| Number of vehicles on BC lane 3
84.	nrVehiclesBCForward3_WeightedMean	0.92	| nr = number at the moment,
85.	nrVehiclesBCForward3_WeightedStDev	0.89	| N = number of observations,
86.	nrVehiclesBCForward3_Min	0.0	| mean, stdev, min, max are weighted stats
87.	nrVehiclesBCForward3_Max	4.0	End statistic #13
88.	nrVehiclesCDForward1_nrVehicles	1.0	Start statistic #14
89.	nrVehiclesCDForward1_N	7199	| Number of vehicles on CD lane 1
90.	nrVehiclesCDForward1_WeightedMean	0.17	| nr = number at the moment,
91.	nrVehiclesCDForward1_WeightedStDev	0.37	| N = number of observations,
92.	nrVehiclesCDForward1_Min	0.0	| mean, stdev, min, max are weighted stats
93.	nrVehiclesCDForward1_Max	1.0	End statistic #14
94.	nrVehiclesCDForward2_nrVehicles	0.0	Start statistic #15
95.	nrVehiclesCDForward2_N	7199	| Number of vehicles on CD lane 2
96.	nrVehiclesCDForward2_WeightedMean	0.29	| nr = number at the moment,
97.	nrVehiclesCDForward2_WeightedStDev	0.45	| N = number of observations,
98.	nrVehiclesCDForward2_Min	0.0	| mean, stdev, min, max are weighted stats
99.	nrVehiclesCDForward2_Max	1.0	End statistic #15
100.	nrVehiclesDEForward1_nrVehicles	21.0	Start statistic #16
101.	nrVehiclesDEForward1_N	7199	| Number of vehicles on DE lane 1
102.	nrVehiclesDEForward1_WeightedMean	14.51	| nr = number at the moment,
103.	nrVehiclesDEForward1_WeightedStDev	4.15	| N = number of observations,
104.	nrVehiclesDEForward1_Min	0.0	| mean, stdev, min, max are weighted stats
105.	nrVehiclesDEForward1_Max	25.0	End statistic #16
106.	nrVehiclesDEForward2_nrVehicles	27.0	Start statistic #17
107.	nrVehiclesDEForward2_N	7199	| Number of vehicles on DE lane 2
108.	nrVehiclesDEForward2_WeightedMean	22.07	| nr = number at the moment,
109.	nrVehiclesDEForward2_WeightedStDev	4.77	| N = number of observations,
110.	nrVehiclesDEForward2_Min	0.0	| mean, stdev, min, max are weighted stats
111.	nrVehiclesDEForward2_Max	33.0	End statistic #17
112.	nrVehiclesF2BForward1_nrVehicles	1.0	Start statistic #18
113.	nrVehiclesF2BForward1_N	7199	| Number of vehicles on F2B lane 1
114.	nrVehiclesF2BForward1_WeightedMean	2.75	| nr = number at the moment,
115.	nrVehiclesF2BForward1_WeightedStDev	1.69	| N = number of observations,
116.	nrVehiclesF2BForward1_Min	0.0	| mean, stdev, min, max are weighted stats
117.	nrVehiclesF2BForward1_Max	9.0	End statistic #18
118.	gtuId	0	GTU-based state generation start - Id# 0-2499
119.	active_0	0	0 = inactive, 1 = active
120.	carTruck_0	NA	car or truck; car = 0; truck = 1
121.	laneId_0	NA	lane of the reference point of the vehicle
122.	lanePos_0	NA	position in meters from the start of the lane
123.	odometer_0	NA	odometer of vehicle in m (driven distance)
124.	speed_0	NA	speed in m/s
125.	acceleration_0	NA	acceleration in m/s2
126.	turnIndicator_0	NA	turnindicator: 0 = off, 1 = left, 2 = right
127.	brakingLights_0	NA	braking lights off = 0, on = 1
128.	distanceHalfSec_0	NA	driven distance in current half second
129.	x_0	NA	x position of vehicle
130.	y_0	NA	y position of vehicle
131.	z_0	NA	z position of vehicle

132.	dirZ_0	NA	z-rotation (orientation) of vehicle in rad.
133.	deltaX_0	NA	deltaX position of vehicle
134.	deltaY_0	NA	deltaY position of vehicle
135.	deltaZ_0	NA	deltaZ position of vehicle
136.	deltaLanePos_0	NA	deltalanePos on the lane of the vehicle
137.	deltaOdometer_0	NA	deltaOdometer of gtu (delta driven distance)
138.	gtuId	1	VEHICLE #1 STARTS HERE
139.	active_1	0	| 20 fields for next vehicle
140.	carTruck_1	NA	|
141.	laneId_1	NA	|
142.	lanePos_1	NA	|
143.	odometer_1	NA	|
144.	speed_1	NA	|
145.	acceleration_1	NA	|
146.	turnIndicator_1	NA	|
147.	brakingLights_1	NA	|
148.	distanceHalfSec_1	NA	|
149.	x_1	NA	|
150.	y_1	NA	|
151.	z_1	NA	|
152.	dirZ_1	NA	|
153.	deltaX_1	NA	|
154.	deltaY_1	NA	|
155.	deltaZ_1	NA	|
156.	deltaLanePos_1	NA	|
157.	deltaOdometer_1	NA	| END OF VEHICLE 1
158.	gtuId	2	VEHICLE #2 STARTS HERE
159.	active_2	0	| 20 fields for next vehicle
160.	carTruck_2	NA	|
161.	laneId_2	NA	|
162.	lanePos_2	NA	|
163.	odometer_2	NA	|
164.	speed_2	NA	|
165.	acceleration_2	NA	|
166.	turnIndicator_2	NA	|
167.	brakingLights_2	NA	|
168.	distanceHalfSec_2	NA	|
169.	x_2	NA	|
170.	y_2	NA	|
171.	z_2	NA	|
172.	dirZ_2	NA	|
173.	deltaX_2	NA	|
174.	deltaY_2	NA	|
175.	deltaZ_2	NA	|
176.	deltaLanePos_2	NA	|
177.	deltaOdometer_2	NA	| END OF VEHICLE 2

C.2. reportState Method Implementation

To generate the state-trace data shown in Table C.1, we implemented the *reportState* method which is scheduled as a method call to be executed by the simulator at every 0.5 seconds delta time (line 97 in Table C.2). The complete code snippet of the *reportState* method is given in Table C.2. In the code snippet, the lines 5-15 are responsible for the reporting of the linkCapacity contextual variables discussed earlier in §5.2.3.1.2. The lines 17-35 reports the transient values of the run statistics for the short-merge model. The lines 37-95 registers the state-variables for each vehicle until all 2,500 vehicle states are

captured in the same row of the state-trace, separated by a tab character. Finally, the reporter completes registering all model variables, moves to a new line and flushes the writer (line 95 and 96) and schedules another method call to be executed at getSimulatorTime()+relativeDuration, where the relativeDuration is 0.5 seconds.

Table C.2. The Java code snippet of the reportState method implemented in the short-merge case study model

```
1   public void reportState() throws Exception
2       {
3         this.out.print(getSimulator().getSimulatorTime().si + "\t0.5");
4
5         double[] ingested = new double[10];
6          ingested[0] = this.nrVehiclesABForward1.ingest(this.net-
7   work.lanes.get("AB.FORWARD1").getGtuList().size());
8          ingested[1] = this.nrVehiclesABForward2.ingest(this.net-
9   work.lanes.get("AB.FORWARD2").getGtuList().size());
10         ingested[2] = this.nrVehiclesBCForward1.ingest(this.net-
11  work.lanes.get("BC.FORWARD1").getGtuList().size());
12         ingested[3] = this.nrVehiclesBCForward2.ingest(this.net-
13  work.lanes.get("BC.FORWARD2").getGtuList().size());
14         ingested[4] = this.nrVehiclesBCForward3.ingest(this.net-
15  work.lanes.get("BC.FORWARD3").getGtuList().size());
16         ingested[5] = this.nrVehiclesCDForward1.ingest(this.net-
17   work.lanes.get("CD.FORWARD1").getGtuList().size());
18         ingested[6] = this.nrVehiclesCDForward2.ingest(this.net-
19   work.lanes.get("CD.FORWARD2").getGtuList().size());
20         ingested[7] = this.nrVehiclesDEForward1.ingest(this.net-
21  work.lanes.get("DE.FORWARD1").getGtuList().size());
22         ingested[8] = this.nrVehiclesDEForward2.ingest(this.net-
23  work.lanes.get("DE.FORWARD2").getGtuList().size());
24         ingested[9] =this.nrVehiclesF2BForward1.ingest(this.net-
25  work.lanes.get("F2B.FORWARD1").getGtuList().size());
26
27            for (SimTally.TimeDoubleUnit tally : this.statTallyArray)
28            {
29               this.out.print("\t" + tally.getN());
30               this.out.print("\t" + tally.getSampleMean());
31               this.out.print("\t" + tally.getSampleStDev());
32               this.out.print("\t" + tally.getMin());
33               this.out.print("\t" + tally.getMax());
34            }
35
```

```
36          int i = 0;
37          for (SimPersistent.TimeDoubleUnit persistent : this.statPersistentArray)
38          {
39              this.out.print("\t" + ingested[i++]);
40              this.out.print("\t" + persistent.getN());
41              this.out.print("\t" + persistent.getWeightedSampleMean());
42              this.out.print("\t" + persistent.getWeightedSampleStDev());
43              this.out.print("\t" + persistent.getMin());
44              this.out.print("\t" + persistent.getMax());
45          }
46
47          for (int gtuId = 0; gtuId < NUM_GTUS; gtuId++)
48          {
49              LaneBasedIndividualGtu gtu = (LaneBasedIndividualGtu) this.network.getGtuByIdMap().get(gtuId)
50              this.out.print("\t" + gtuId);
51              this.out.print("\t" + (gtu == null ? "0" : "1"));
52              this.out.print("\t" + (gtu == null ? "NA" : gtu.getGTUType().getId().equals("car") ? "0" : "1"));
53              this.out.print("\t" + (gtu == null ? "NA" : gtu.getReferencePosition().getLane().getFullId()));
54              this.out.print("\t" + (gtu == null ? "NA" : gtu.getReferencePosition().getPosition().si));
55              this.out.print("\t" + (gtu == null ? "NA" : gtu.getOdometer().si));
56              this.out.print("\t" + (gtu == null ? "NA" : gtu.getSpeed().si));
57              this.out.print("\t" + (gtu == null ? "NA" : gtu.getAcceleration().si));
58              this.out.print("\t" + (gtu == null ? "NA" : gtu.getTurnIndicatorStatus().ordinal()));
59              this.out.print("\t" + (gtu == null ? "NA" : gtu.isBrakingLightsOn() ? "1" : "0"));
60              this.out.print("\t" + (gtu == null ? "NA" : gtu.getOperationalPlan().getTotalLength().si));
61              OrientedPoint3d pos = gtu == null ? null : gtu.getLocation();
62              this.out.print("\t" + (gtu == null ? "NA" : pos.x));
63              this.out.print("\t" + (gtu == null ?  "NA" : pos.y));
64              this.out.print("\t" + (gtu == null ? "NA" : pos.z));
65              this.out.print("\t" + (gtu == null ?  "NA": pos.getDirZ()));
66              LastGtuRecord last = this.network.getLastGtu(gtuId);
67              String dx = last != null ? String.valueOf(pos.x - last.x) : "NA";
68              this.out.print("\t" + dx);
69              String dy = last != null ? String.valueOf(pos.y - last.y) : "NA";
70              this.out.print("\t" + dy);
71              String dz = last != null ? String.valueOf(pos.z - last.z) : "NA";
72              this.out.print("\t" + dz);
73              String dLP = "NA";
74              if (last != null)
75              {
76                  try
77                  {
78                      if (last.firstLane.length() == 0)
79                      {
80                          last.firstLane = gtu.getReferencePosition().getLane().getFullId();
```

```
81                      }
82                      if (gtu.getReferencePosition().getLane().getFullId().equals(last.lastLane))
83                      {
84                          dLP = String.valueOf(gtu.getReferencePosition().getPosition().si - last.lastLaneX);
85                      }
86                  }
87                  catch (GtuException exception)
88                  {
89                      exception.printStackTrace();
90                  }
91              }
92              this.out.print("\t" + dLP);
93              String dOdo = last != null ? String.valueOf(gtu.getOdometer().si - last.odo) : "NA";
94              this.out.print("\t" + dOdo);
95              if (gtu != null && last != null)
96              {
97                  last.x = pos.x;
98                  last.y = pos.y;
99                  last.z = pos.z;
100                 last.lastLane = gtu.getReferencePosition().getLane().getFullId();
101                 last.lastLaneX = gtu.getReferencePosition().getPosition().si;
102                 last.odo = gtu.getOdometer().si;
103             }
104         }
105         this.out.println();
106         this.out.flush();
107         getSimulator().scheduleEventRel(Duration.instantiateSI(0.5), this, this, "reportState", null);
108     }
```

SUMMARY / SAMENVATTING

Summary

Large-scale complex systems are characterized by a large number of interconnected variables and a diverse set of interactions. As the demand for the development and optimization of large-scale systems is growing, so does the need for better techniques to understand their underlying dynamic behavior and predict and manage their long-term performance. With the increased capabilities of computer technology, we have been able to run simulation models for these systems that are larger in scale and higher in complexity. While these advancements have enabled more accurate representations of real-world systems, the ever-increasing scale and complexity of simulation models may eventually result in models that are too complex to work with – giving rise to *large-scale complex simulation models*.

Large-scale complex simulation models raise new questions and challenges for the modeling and simulation community, including how models of such large scale and complexity can be expressed and modeled more efficiently, how to ensure that their representations of the underlying systems' complex dynamic behavior are valid, and what tools and techniques can be used to do so. Model abstraction is a strategy for dealing with the scale and complexity of large-scale complex simulation models. However, traditional model abstraction methods such as metamodeling and multiresolution modeling are limited in their ability to automate the model abstraction process.

A more viable approach for automating the model abstraction and preserving the underlying dynamic behavior for large-scale complex simulation models, specifically discrete-event simulation models, may be model abstraction at the transformation level (also referred to as generative system level or the state transition level). Abstraction at the transformation level can be achieved using state-trace data that encapsulate the model's dynamic behavior. State-trace data describe the dynamic behavior of the model at discrete points in time, allowing modelers to capture the state-transitions of the model over the runtime. The state-trace data can be expressed as event sequences or multivariate time series consisting of categorical variables, numerical variables, or both. As such, state-trace data can be used to discover behavioral patterns relevant to the desired level of

abstraction. Assuming a morphism relation between a pair of system specifications at the state-transition level, a valid correspondence relation (mapping) can be established between the base model's (i.e., the more detailed system specification) and the lumped model's (i.e., the abstracted system specification) state-transitions, where the lumped model uses these previously discovered behavioral patterns as aggregated states.

However, traces of state-transitions obtained from the runs of large-scale complex simulation models can get extensive in terms of their volume (the size of the trace data and the number of state variables to be sampled from different model components) and variety (the number of unique states). This consequently confines modelers' ability to identify and utilize state trace patterns for model abstraction. Data mining and machine learning methods have been designed to ease the process of discovering frequent patterns in temporal data. Although such methods have proven to be useful for recognizing behavioral patterns within large volumes of data, they have not yet been applied to automate the abstraction of large-scale complex discrete-event simulation models. This would require techniques that not only identify important behavioral patterns in state-trace data, but also generate aggregated states at various abstraction levels to construct models at a higher level of abstraction. Integrating the fields of modelling and simulation and temporal data mining may provide a promising direction to deal with the automated abstraction of large-scale complex simulation models.

In this dissertation, we aim to investigate to what extent the abstraction of large-scale complex simulation models, specifically the discrete-event simulation models expressed in DEVS formalism, can be automated using their state-trace data. In order to achieve this objective, we designed a method that integrates the fields of modeling and simulation and temporal data mining by utilizing state-trace data and applying frequent episode mining techniques to discover behavioral patterns. We demonstrated the practical application of our method using three simulation case studies with increasing scale and complexity and with different model characteristics.

In DEVS, the next state can be determined solely by knowing the current state and the time elapsed (i.e., memoryless). Consequently, the dynamic behavior of a DEVS model encapsulated in state-trace data can be simplified by applying frequent episode mining algorithms to identify frequent state-trace record pairs corresponding to the model's

state-transitions. The resulting simplified model can subsequently be formally described using Markov Chain. The memoryless property of Markov Chain states is akin to the discrete-event model states in DEVS and the state-transition probabilities represented in matrix format is the transition matrix of a valid stochastic discrete-time Markov Chain. To determine the best performing frequent episode mining algorithm for the state-trace data generated by the execution of discrete-event simulation models, we compared several popular frequent episode mining algorithms, such as MINEPI, MINEPI+, and EMMA. Experiment results (Appendix A) revealed that, at the time of our research, EMMA is the fastest frequent episode mining algorithm in finding all unique state-trace records and consecutive state-trace record pairs from state-trace data of varying lengths.

In Chapter 3, we first provided a formal description of discrete-event simulation generated state-trace data and state-trace record, which is essential for a unified understanding of these concepts and to properly position our method in the modeling and simulation literature. Subsequently, we presented a breakdown of our method and discussed a range of considerations at each step that are essential for generating a valid abstraction of the base model. In essence, the quality of the behavioral patterns discovered by the temporal data mining techniques, and therefore, the success of our method to automate the generation of valid model abstractions, is highly dependent on the state-trace data generated by the base models – the large-scale complex discrete-event simulation models. Therefore, the first step of our proposed method is to addresses the considerations and actions regarding the generation of state-trace data from the discrete-event simulations. We discussed that there are several factors the modeler needs to consider before deciding on the content of the state-trace data and its generation from the simulation of discrete-event simulation models: *representation of time*, *type of model (stochastic vs. deterministic)*, *type of simulation (terminating vs. non-terminating)*, *inclusion of input data*, *inclusion of output data,* and *complete vs. partial model state-trace data*. In Chapter 3, we closely followed the guidelines provided for each consideration by our method and demonstrated the step-by-step generation of the data using a simple M/M/1 single-server queueing model. Similarly, we generated state-trace data from the simulation of the battlefield model in Chapter 4 and the traffic model in Chapter 5.

After the generation of state-trace data from the base model, the next step of our method is the application of the frequent episode mining algorithm for the task of discovering serial state-transitions and generating the transition probability matrix *P* of the discrete-time Markov Chain. We identified several considerations regarding the application of the temporal data mining tasks to the state-trace data such as the *preprocessing of the state-trace data*, *the application of the EMMA algorithm*, and *the construction of the transition matrix*. For the preprocessing of the state-trace data, we provided guidance to the modelers on several important sub-considerations such as how to address the input format requirements of the EMMA algorithm, how to apply different quantization techniques to deal with a large range of values for state variables in the state trace, and how to format the state-trace data when performing model abstraction at the system state level and at the model component level. After preprocessing the state-trace data, the EMMA algorithm is applied to construct the transition matrices represented as sparse matrices and generate the Markov Chains. For the M/M/1 and battlefield case studies, we demonstrated the application of EMMA algorithm and the generation of Markov Chains at system-state level. For the larger and more complex traffic case study (in Chapter 5), we introduced a skip factor variable to the EMMA algorithm by means of a preprocessing transformation to the state-trace data to identify transition probabilities at model-component level, and thus, construct unique transition matrices for each individual vehicle in the traffic system.

After obtaining the Markov Chain, the next step in our method is to design Markov Chain simulation experiments with multiple runs (using unique Random Number Generator seeds for each run) and to generate state-trace data from these runs. We demonstrated the difference between simulating the Markov Chain of terminating base models and non-terminating base models. In a non-terminating Markov Chain (Chapters 3 and 5), similar to a non-terminating discrete-event simulation, an artificial end-condition should be introduced to terminate the run of the Markov Chain model. However, for the terminating battlefield model in Chapter 4, we demonstrated that the accuracy of the represented terminating behavior of the base model can be problematic (i.e., models may not converge towards the end state) in the absence of absolute time as a state variable in the state-trace data. On the other hand, the representation of absolute simulation time would introduce monotonously increasing values in the state-trace data, which will

undermine the ability of the frequent episode mining algorithms to find recurring patterns. To overcome this issue and accurately represent the terminating behavior, contextual variables that provide the progression over time should be included, albeit without adding monotonously increasing values to each state record.

Once the state-trace data are generated from the Markov Chain, the final step of our proposed method is to evaluate the accuracy and validity of the abstracted Markov Chain. To do so, the Markov Chain generated state-trace data is compared with an independent state-trace validation set obtained from the base model. In this dissertation, key statistics and performance measures obtained from the original model's validation sets and the abstracted Markov Chains-generated state-trace data were compared using descriptive (e.g., spread measures, histograms, box and whisker plots) and inferential statistics (e.g., χ^2-test, Student's *t*-test, two-sample Kolmogorov-Smirnov). The results obtained from the three case studies, and specifically their validation process, demonstrated that our method is able to obtain Markov Chain estimates of performance measures with an acceptable level of precision that do not significantly deviate from and follow a similar distribution as the original model's validation set. We also argued that it is of great importance for the modeler to not solely rely on *p*-values to judge the accuracy of the base model's representation but also evaluate descriptive statistics as well as the magnitude of the difference, that is, Markov Chains' estimates should be considered within the context of the system of interest and the goals of the model abstraction study. For instance, in the traffic case study (Chapter 5), the Student's *t*-test results indicated that the car and truck mean speeds generated from the Markov Chain differed significantly from the mean speeds obtained from the base model for Route F; however, the absolute mean difference was insignificantly small given the nature of the case study. Similarly, the results of the χ^2-test in Chapter 3 illustrated that a large sample size may result in a significant difference between the two distributions (a known problem with significance testing based on *p*-values), even though the Cramer's *V* effect size and graphical visualizations suggested that the magnitude of this difference was very small.

Finally, the results in this dissertation revealed that increasing the model's resolution by increasing the quantization levels of the variables (e.g., the number of bins) alone does not improve the accuracy of the model in the absence of relevant contextual variables.

In fact, the results of the battlefield case study (Chapter 4) showed that the presence of contextual variables is important for the accurate representation of the terminating behavior of the base model, and specifically, for obtaining run-lengths that are sufficiently close to the base model's run-length. Similarly, in the traffic case study (Chapter 5), our validation study highlighted how essential the contextual variables are for the accuracy of the Markov Chain when representing the stochastic behavior (e.g., the generation of the traffic stream) of the base model. Experiments indicated that the average number of vehicles generated (across all runs) by the Markov Chains with the additional contextual variables got closer to the base model's numbers than the Markov Chains without those additional contextual variables.

In sum, the presented research showed that our novel method is capable of automating the abstraction of large-scale and complex discrete-event simulation models with large state-spaces. Although the process can be fully automated, some decisions, such as which state variables and contextual variables to be included and what quantization levels to be used, will benefit from support by modelers who understand the simulation model and the context in which the model is to be used. The findings throughout this dissertation demonstrated that the method is able to generate valid Markov Chains from the base model's state-trace data that adequately represent the dynamic behavior of the base model and estimate its key performance measures with sufficient precision and accuracy. Moreover, by presenting a sequence of clear steps and decisions for our method and addressing a range of considerations that modelers may face, we demonstrated how state-trace data and temporal data mining techniques may best be applied to automatically abstract simulation models with different characteristics.

Samenvatting

Grootschalige complexe systemen worden gekenmerkt door een groot aantal variabelen en hun diverse relaties en interacties. Naarmate de vraag naar de ontwikkeling en optimalisatie van grootschalige systemen toeneemt, groeit ook de behoefte aan betere technieken om het onderliggende dynamische gedrag en de langetermijnprestaties te kunnen begrijpen, voorspellen en beïnvloeden. Vanwege hun omvang en complexiteit is het echter vaak te moeilijk of te duur om rechtstreeks met grootschalige complexe systemen te experimenteren. Derhalve zijn modellen nodig die de complexiteit van deze systemen kunnen afbeelden, maar die wel praktisch zijn om mee te werken. Een bruikbare en effectieve methode om grootschalige complexe systemen te kunnen analyseren en evalueren is *simulatie*. Dankzij de toegenomen mogelijkheden van de computertechnologie zijn we in staat om simulatiemodellen van steeds grotere schaal en hogere complexiteit te ontwikkelen. Hoewel hierdoor veel vooruitgang is geboekt in het nauwkeurig representeren van *real-world* systemen, kan de alsmaar groeiende schaal en complexiteit van zowel de systemen als hun simulatiemodellen ertoe leiden dat uiteindelijk ook de simulatiemodellen zelf te complex worden om mee te werken – resulterend in *grootschalige complexe simulatiemodellen*.

Grootschalige complexe simulatiemodellen werpen nieuwe vragen en uitdagingen op voor modelleurs en simulatiedeskundigen, zoals de vraag hoe modellen van een dergelijke grote schaal en complexiteit op een effectieve en efficiënte wijze kunnen worden afgebeeld, hoe kan worden gewaarborgd dat de representatie van het complexe dynamische gedrag van de onderliggende systemen valide is, en welke instrumenten en technieken hiervoor kunnen worden gebruikt. Een van de strategieën om met grootschalige complexe simulatiemodellen om te gaan is het abstraheren van modellen (*model abstraction*). Traditionele modelabstractiemethoden (bijvoorbeeld metamodellering en multiresolutiemodellering) zijn echter beperkt in hun mogelijkheden om het abstractieproces te automatiseren, terwijl handmatige abstractie van grootschalige complexe simulatiemodellen op structuurniveau – zoals de meeste conventionele modelabstractiemethoden

doen – vaak niet haalbaar is gezien het grote en diverse aantal modelcomponenten en hun onderlinge relaties.

Een wellicht meer haalbare benadering voor het automatisch abstraheren van grootschalige complexe simulatiemodellen, met name *discrete-event* simulatiemodellen, is modelabstractie op niveau van de transformaties (theoretisch overeenkomstig met het generatief systeemniveau dat zich richt op de toestandsovergangen of *state-transitions*). Abstractie op transformatieniveau kan worden gerealiseerd door gebruik te maken van *state-trace* data die het dynamische gedrag van het model representeren. *State-trace* data beschrijven de opeenvolgende toestandsovergangen van het model op discrete tijdstippen, waardoor modelleurs de geschiedenis van een simulatierun kunnen vastleggen en inzicht krijgen in hoe complexe verschijnselen zich in de tijd ontwikkelen. *State-trace* data kunnen worden uitgedrukt als reeksen van gebeurtenissen (*event sequences*) of multivariate tijdreeksen bestaande uit categorische variabelen, numerieke variabelen of een combinatie daarvan. Zodoende kunnen *state-trace* data worden gebruikt om gedragspatronen te ontdekken die relevant zijn voor het gewenste abstractieniveau. Uitgaande van een morfisme tussen twee systeemspecificaties op het transformatieniveau, kan een valide relatie (*mapping*) worden bepaald tussen de toestandsovergangen van het basismodel (d.w.z. de meer gedetailleerde systeemspecificatie) en die van het geclusterde model (d.w.z. de geabstraheerde systeemspecificatie), waarbij het geclusterde model deze eerder ontdekte gedragspatronen als geaggregeerde toestanden gebruikt.

Een beperking in het gebruik van *state-trace* data voor modelabstractie is dat *state-trace* data die verkregen zijn uit de runs van grootschalige complexe simulatiemodellen, zeer omvangrijk kunnen worden in zowel volume (de omvang van de *trace* data en het aantal toestandsvariabelen van verschillende modelcomponenten) als variëteit (het aantal unieke toestanden). Dit maakt het lastig voor modelleurs om frequente patronen in deze data te identificeren en te gebruiken voor modelabstractie. Datamining- en machine learning-methoden zijn ontworpen om het ontdekken van frequente patronen in temporele datasets te vergemakkelijken. Hoewel dergelijke methoden nuttig zijn gebleken voor het herkennen van patronen in grote hoeveelheden data, zijn ze tot op heden nog niet toegepast voor het automatiseren van de abstractie van grootschalige complexe *discrete-event* simulatiemodellen. Dit zou technieken vereisen die niet alleen belangrijke

gedragspatronen in *state-trace* data identificeren, maar ook de geaggregeerde toestanden kunnen genereren om modellen op een hoger abstractieniveau te construeren. Het integreren van concepten en technieken vanuit de simulatie en de temporele datamining kan daarom een veelbelovende oplossingsrichting bieden voor het automatiseren van de abstractie van grootschalige complexe simulatiemodellen.

In dit proefschrift onderzoeken we in hoeverre de abstractie van grootschalige complexe simulatiemodellen, specifiek *discrete-event* simulatiemodellen beschreven volgens het DEVS-formalisme, kan worden geautomatiseerd met behulp van hun *state-trace* data. Hiertoe hebben we een methode ontwikkeld waarbij *frequent episode mining*-technieken worden toepast op *state-trace* data met het doel gedragspatronen te ontdekken. We demonstreren de praktische toepasbaarheid van onze methode aan de hand van drie casestudies met simulatiemodellen van toenemende omvang en complexiteit en met verschillende eigenschappen.

Kenmerkend voor DEVS is dat toekomstige toestanden enkel afhangen van de huidige toestand en de verstreken tijd (m.a.w., er is geen "geheugen" nodig). Dit betekent dat het dynamisch gedrag van een DEVS-model, samengevat in de *state-trace* data, vereenvoudigd kan worden door *frequent episode mining*-algoritmen toe te passen op opeenvolgende paren van toestandsovergangen van het model. Het resulterende vereenvoudigde model kan vervolgens formeel worden beschreven met een Markov Chain. De geheugenloze eigenschap van Markov Chain toestanden komt overeen met die van de *discrete-event* toestanden in een DEVS-model. De matrix met overgangskansen van opeenvolgende toestanden van een DEVS-model vormt de basis voor de overgangsmatrix van een stochastische *discrete-time* Markov Chain.

Om te bepalen welk *frequent episode mining* algoritme het beste presteert voor het identificeren van frequente toestandsovergangen in de *state-trace* data van *discrete-event* simulatiemodellen, hebben we verschillende populaire algoritmen vergeleken, waaronder MINEPI, MINEPI+ en EMMA. Uit de resultaten van deze vergelijking (zie Appendix A) blijkt dat het EMMA-algoritme ten tijde van het onderzoek het snelst was in het vinden van alle unieke *state-trace records* en opeenvolgende *state-trace record*-paren uit *state-trace* datasets met verschillende omvang.

In Hoofdstuk 3 hebben we eerst een formele beschrijving gepresenteerd van door *discrete-event* simulatie gegenereerde *state-trace* data en *state-trace records*, daar dit essentieel is voor een eenduidig begrip van deze concepten en voor het correct positioneren van onze methode in de simulatieliteratuur. Vervolgens hebben we onze methode stap-voor-stap uiteengezet en een reeks overwegingen besproken die essentieel zijn voor het genereren van een valide abstractie van het basismodel.

In wezen is de kwaliteit van de gedragspatronen die worden ontdekt door toepassing van de temporele dataminingtechnieken, en daarmee het succes van onze methode om het abstraheren van grootschalige complexe simulatiemodellen te automatiseren, in hoge mate afhankelijk van de *state-trace* data die door de basismodellen worden gegenereerd. De eerste stap in onze methode is dan ook om een aantal overwegingen en keuzes te bespreken met betrekking tot het genereren van *state-trace* data uit de *discrete-event* simulatie. Er zijn verschillende factoren waarmee de modelleur rekening moet houden voordat hij/zij een beslissing neemt over de inhoud van de *state-trace* data en de manier waarop deze worden gegenereerd. Belangrijke factoren en keuzes zijn de weergave van tijd, het type model (stochastisch versus deterministisch), het type simulatie (eindigend versus niet-eindigend), het meenemen van input data, het meenemen van output data, en het abstraheren van de volledige *state-trace* data of van een subset. In Hoofdstuk 3 hebben we de richtlijnen voor elk van deze overwegingen secuur beschreven en de abstractie van een simulatiemodel op basis van de *state-trace* data gedemonstreerd aan de hand van een eenvoudig M/M/1 single-server wachtrijmodel. Op eenzelfde wijze hebben we *state-trace* data gegenereerd uit de simulatie van een militaire oefening in Hoofdstuk 4 en een verkeersmodel in Hoofdstuk 5.

De volgende stap in onze methode is de toepassing van het *frequent episode mining*-algoritme op de uit het basismodel verkregen *state-trace* data om opeenvolgende toestandsovergangen te identificeren en een overgangsmatrix *P* van de *discrete-time* Markov Chain te genereren. Voor deze stap hebben we diverse aandachtspunten geïdentificeerd voor de voorbewerking van de *state-trace* data, de toepassing van het EMMA-algoritme en de constructie van de overgangsmatrix. Voor de voorbewerking van de *state-trace* data zijn richtlijnen beschreven voor de eisen aan de invoer voor het EMMA-algoritme, het toepassen van verschillende kwantisatietechnieken wanneer de toestandsvariabelen een

groot waardenbereik hebben, en het afbeelden van de *state-trace* data voor het uitvoeren van abstractie op systeemniveau versus abstractie op modelcomponentniveau. Na het voorbewerken van de *state-trace* data kan het EMMA-algoritme worden toegepast om een overgangsmatrix te genereren en een Markov Chain te construeren. De M/M/1 (Hoofdstuk 3) en militaire (Hoofdstuk 4) casestudies demonstreren de toepassing van het EMMA-algoritme en het genereren van Markov Chains op systeemniveau. Voor het grotere en complexere verkeersmodel (Hoofdstuk 5) hebben we via voorbewerking van de *state-trace* data eerst een zogenaamde *skip-factor* in het EMMA-algoritme geïntroduceerd om overgangskansen op modelcomponentniveau te kunnen identificeren, en zodoende unieke overgangsmatrices te construeren voor elk afzonderlijk voertuig in het verkeerssysteem.

Na het construeren van de Markov Chain is de volgende stap in de methode het ontwerpen van simulatie-experimenten met het Markov Chain model die bestaan uit meerdere runs (gebruikmakend van unieke seeds voor de toevalsgetalgenerator voor iedere run), en het genereren van *state-trace* data uit deze runs. In dit proefschrift hebben we het verschil laten zien tussen het simuleren van Markov Chains van eindigende basismodellen en niet-eindigende basismodellen. Voor een niet-eindigende Markov Chain (Hoofdstukken 3 en 5), vergelijkbaar met een niet-eindigende discrete-event simulatie, moet een kunstmatige eindconditie worden geïntroduceerd om de run van de Markov Chain te beëindigen. In Hoofdstuk 4, waar we onze methode demonstreren aan de hand van een eindigend model voor een militaire oefening, hebben we laten zien dat de nauwkeurigheid van het terminerend gedrag van een basismodel problematisch kan zijn (d.w.z. dat modellen mogelijk niet convergeren naar de eindtoestand) als absolute tijd niet wordt meegenomen als toestandsvariabele in de *state-trace* data. Helaas zou het opnemen van absolute simulatietijd monotoon oplopende waarden in de *state-trace* data introduceren, wat het vermogen van het *frequent episode mining*-algoritme om terugkerende patronen te vinden ondermijnt. Om dit probleem op te lossen en het eindigende gedrag nauwkeurig te representeren, moeten contextuele variabelen worden opgenomen die de progressie in tijd weergeven, echter zonder een reeks monotoon oplopende waarden voor elk state record te introduceren.

Wanneer de *state-trace* data van de Markov Chain modelruns zijn verkregen, is de laatste stap in de methode het evalueren van de nauwkeurigheid en validiteit van de geabstraheerde Markov Chain. Hiertoe worden *state-trace* data die gegenereerd zijn uit de Markov Chain vergeleken met een onafhankelijke *state-trace* validatieset verkregen uit het basismodel. In dit proefschrift hebben we statistieken en prestatie-indicatoren van beide modellen (de set afkomstig uit de geabstraheerde Markov Chain en de validatieset van het basismodel) vergeleken aan de hand van beschrijvende statistische technieken (bijv. spreidingsmaten, histogrammen, boxplots) en inferentiële statistische technieken (bijv. χ^2-test, Student's *t*-toets, Kolmogorov-Smirnov toets). De resultaten van de drie casestudies, en specifiek hun validatieproces, toonden aan dat onze methode in staat is om met een aanvaardbaar precisieniveau Markov Chain-schattingen van prestatie-indicatoren te genereren die niet significant afwijken van, en een vergelijkbare verdeling volgen als, de validatieset van het basismodel. Het onderzoek geeft aan dat het van groot belang is dat de modelleur niet alleen op *p*-waarden vertrouwt om de nauwkeurigheid van het geabstraheerde model te beoordelen, maar daarnaast ook beschrijvende statistieken en effectgroottes evalueert om te beoordelen hoe betekenisvol eventuele afwijkingen zijn. Met andere woorden, de schattingen van de Markov Chain moeten altijd worden beschouwd binnen de context van het gesimuleerde systeem en de doelen van de abstractiestudie. In het verkeersmodel (Hoofdstuk 5) lieten de resultaten van de *t*-toets bijvoorbeeld zien dat de gemiddelde snelheden voor voertuigen op Route F verkregen uit de Markov Chain significant verschilden van de gemiddelde snelheden verkregen uit het basismodel. Echter, het absolute gemiddelde verschil was onbeduidend klein. Evenzo illustreerden de resultaten van de χ^2-test in Hoofdstuk 3 hoe een grote steekproefomvang kan leiden tot een significant verschil tussen de twee distributies (een bekend probleem met significantietoetsen op basis van *p*-waarden), ondanks dat de Cramer's *V*-effectgrootte en grafische visualisaties aangaven dat de omvang van dit verschil erg klein was.

Tenslotte lieten de bevindingen in dit proefschrift zien dat het verhogen van de resolutie van het model door enkel een hoger kwantisatieniveau (bijvoorbeeld een groter aantal bins) toe te passen niet resulteert in een verbeterde nauwkeurigheid wanneer relevante contextuele variabelen in het model ontbreken. In de militaire casestudie (Hoofdstuk 4) bleek dat de aanwezigheid van contextuele variabelen belangrijk is voor een accurate

representatie van het terminerend gedrag van het basismodel, in het bijzonder voor het verkrijgen van runlengten die voldoende overeenkomen met de runlengten van het basismodel. Op eenzelfde wijze liet de verkeerscasestudie (Hoofdstuk 5) zien hoe essentieel contextuele variabelen zijn voor het nauwkeurig abstraheren van stochastisch gedrag in het basismodel (bijv. de generatie van de verkeersstroom). Experimenten toonden aan dat schattingen van het aantal voertuigen gegenereerd door Markov Chains met extra contextuele variabelen dichterbij die van het basismodel lagen dan schattingen van Markov Chains zonder deze extra contextuele variabelen.

Samenvattend heeft het onderzoek gepresenteerd in dit proefschrift laten zien dat onze nieuwe methode in staat is om de abstractie van grootschalige en complexe discrete-event simulatiemodellen met grote toestandsruimten te automatiseren. Hoewel het proces volledig kan worden geautomatiseerd, zullen sommige beslissingen, zoals welke toestandsvariabelen en contextuele variabelen moeten worden opgenomen en welke kwantisatieniveaus moeten worden gebruikt, baat hebben bij ondersteuning door modelleurs die het simulatiemodel en de context waarin het model moet worden gebruikt begrijpen. De bevindingen in dit proefschrift hebben aangetoond dat de methode in staat is om uit de *state-trace* data van het basismodel valide Markov Chains te genereren die het dynamisch gedrag van het basismodel adequaat weergeven en de belangrijkste prestatie-indicatoren van het basismodel met voldoende precisie en nauwkeurigheid schatten. Door de methode te structureren volgens een helder stappenplan en een reeks overwegingen te behandelen waarmee modelleurs te maken kunnen krijgen, hebben we bovendien aangetoond hoe *state-trace* data en temporele dataminingtechnieken het best kunnen worden toegepast om simulatiemodellen met verschillende karakteristieken automatisch te abstraheren.

References

References

Ackoff, R. L., & Emery, F. E. (1972). *On purposeful systems*. Aldine-Atherton.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), *Proceedings of the 20th International Conference on Very Large Data Bases* (pp. 487–499). https://dl.acm.org/doi/10.5555/645920.672836

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering*, 3–14. https://doi.org/10.1109/ICDE.1995.380415

Alavi, M., Carlson, P., & Brooke, G. (1989). The ecology of MIS Research: A twenty year status review. In J. I. DeGross, J. C. Henderson, & B. R. Konsynski (Eds.), *Proceedings of the Tenth International Conference on Information Systems* (pp. 363–375). https://doi.org/10.1145/75034.75065

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communincations of the ACM, 26*(11), 832–843. https://doi.org/10.1145/182.358434

Anderson, P. W. (1972). More is different. *Science, 177*(4047), 393. https://doi.org/10.1126/science.177.4047.393

Anderson, T. W., & Goodman, L. A. (1957). Statistical Inference about Markov Chains. *The Annals of Mathematical Statistics, 28*(1), 89-110. https://doi.org/10.1214/aoms/1177707039

Ao, X., Luo, P., Li, C., Zhuang, F., & He, Q. (2015). Online frequent episode mining. *2015 International Conference on Data Engineering*, 891–902. IEEE. https://doi.org/10.1109/ICDE.2015.7113342

Arthur, J., Sargent, R., Dabney, J., Law, A., & Morrison, J. D. (1999). Verification and validation: What impact should project size and complexity have on attendant V&V activities and supporting infrastructure? In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, & G. W. Evans (Eds.), *1999 Winter Simulation Conference Proceedings* (pp. 148–155). IEEE. https://doi.org/10.1109/WSC.1999.823064

Astrup, R., Coates, K. D., & Hall, E. (2008). Finding the appropriate level of complexity for a simulation model: An example with a forest growth model. *Forest Ecology and Management*, *256*(10), 1659–1665. https://doi.org/10.1016/j.foreco.2008.07.016

Atluri, G., Karpatne, A., & Kumar, V. (2018). Spatio-temporal data mining: A survey of problems and methods. *ACM Computing Surveys, 51*(4), 1–41. https://doi.org/10.1145/3161602

Balci, O. (1994). Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Annals of Operations Research, 53*(1), 121–173. https://doi.org/10.1007/BF02136828

Balci, O. (1997). Verification validation and accreditation of simulation models. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *1997 Winter Simulation Conference Proceedings* (pp. 135–141). https://doi.org/10.1145/268437.268462

Balci, O., Ball, G. L., Morse, K. L., Page, E., Petty, M. D., Tolk, A., & Veautour, S. N. (2017). Model reuse, composition, and adaptation. In R. Fujimoto, C. Bock, W. Chen, E. Page, & J. H. Panchal (Eds.), *Research challenges in modeling and simulation for engineering complex systems* (pp. 87–115). Springer. https://doi.org/10.1007/978-3-319-58544-4_6

Balci, O., Ormby, W. F., Carr, J. T., & Saadi, S. D. (2000). Planning for verification, validation, and accreditation of modeling and simulation applications. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *2000 Winter Simulation Conference Proceedings* (pp. 829–839). IEEE. https://doi.org/10.1109/WSC.2000.899881

Banks, J. (1998). Principles of simulation. In J. Banks (Ed.), *Handbook of simulation* (pp. 1–30). John Wiley & Sons. https://doi.org/10.1002/9780470172445.ch1

Banks, J., & Carson, J. S. (1984). *Discrete-event system simulation*. Prentice-Hall.

Banks, J., Carson, J. S., Nelson, B. L., & Nicol, D. (2010). *Discrete-event system simulation* (5th ed.). Pearson.

Barton, R. R. (1992). Metamodels for simulation input-output relations. In J. J. Swain, D. Goldsman, R. C. Crain, & J. R. Wilson (Eds.), *1992 Winter Simulation Conference Proceedings* (pp. 289–299). https://doi.org/10.1145/167293.167352

Barton, R. R. (2015). Tutorial: Simulation metamodeling. In L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, & M. D. Rossetti (Eds.), *2015 Winter Simulation Conference Proceedings* (pp. 1765–1779). https://doi.org/10.1109/wsc.2015.7408294

Benabdellah, A. C., Benghabrit, A., & Bouhaddou, I. (2019). A survey of clustering algorithms for an industrial context. *Procedia Computer Science, 148*, 291-302. Elsevier. https://doi.org/10.1016/j.procs.2019.01.022

Berkhin, P. (2006). A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, & M. Teboulle (Eds.), *Grouping multidimensional data: Recent advances in clustering* (pp. 25–71). Springer. https://doi.org/10.1007/3-540-28349-8_2

Bhat, U. N. (2008). Simple Markovian queueing systems. In U. N. Bhat (Ed.), *An introduction to Queueing Theory* (pp. 37–83). Birkhäuser Boston. https://doi.org/10.1007/978-0-8176-4725-4_4

Biller, B., & Gunes, C. (2010). Introduction to simulation input modeling. In B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, & E. Yücesan (Eds.), *2010 Winter Simulation Conference Proceedings* (pp. 49–58). https://dl.acm.org/doi/10.5555/2433508.2433517

Birta, L. G., & Özmizrak, F. N. (1996). A knowledge-based approach for the validation of simulation models: The foundation. *ACM Transactions on Modeling and Computer Simulation, 6*(1), 76–98. https://doi.org/10.1145/229493.229511

Boardman, J., & Sauser, B. (2008). *Systems Thinking: Coping with 21st century problems*. CRC Press.

Buyya, R., Broberg, J., & Goscinski, A. (2011). *Cloud computing: Principles and paradigms*. Wiley. https://doi.org/10.1002/9780470940105

Carothers, C., Ferscha, A., Fujimoto, R., Jefferson, D., Loper, M., Marathe, M., Mosterman, P., Taylor, S. J. E., & Vakilzadian, H. (2017). Computational Challenges in Modeling and Simulation. In R. Fujimoto, C. Bock, W. Chen, E. Page, & J. H. Panchal (Eds.), *Research challenges in modeling and simulation for engineering complex systems* (pp. 45–74). Springer. https://doi.org/10.1007/978-3-319-58544-4_4

Cash, J., & Nunamaker, J. (1989). *The information systems research challenge, Vol I: Qualitative research methods*. Harvard Business School Press.

Cash, J., & Nunamaker, J. (1990). *The information systems research challenge, Vol II: Qualitative research methods*. Harvard Business School Press.

Cash, J., & Nunamaker, J. (1991). *The information systems research challenge, Vol III: Qualitative research methods*. Harvard Business School Press.

Castro, R., Kofman, E., & Wainer, G. (2010). A formal framework for stochastic discrete event system specification modeling and simulation. *SIMULATION, 86*(10), 587–611. https://doi.org/10.1177/0037549709104482

Cellier, F. E. (1991). General system problem solving paradigm for qualitative modeling. In P. A. Fishwick & P. A. Luker (Eds.), *Qualitative simulation modeling and analysis* (pp. 51–71). Springer. https://doi.org/10.1007/978-1-4613-9072-5_3

Cha, S., & Srihari, S. N. (2002). On measuring the distance between histograms. *Pattern Recognition, 35*(6), 1355–1370. https://doi.org/10.1016/S0031-3203(01)00118-2

Chapela-Campa, D., Mucientes, M., & Lama, M. (2019). Mining frequent patterns in process models. *Information Sciences, 472*, 235–257.

Chen, W., & Hirschheim, R. (2004). A paradigmatic and methodological examination of information systems research from 1991 to 2001. *Information Systems Journal, 14*(3), 197–235. https://doi.org/10.1111/j.1365-2575.2004.00173.x

Chow, A. C. H., & Zeigler, B. P. (1994). Parallel DEVS: A parallel, hierarchical, modular, modeling formalism. In J. D. Tew, S. Manivannan, D. A. Sadowski, & A. F. Seila (Eds.), *1994 Winter Simulation Conference Proceedings* (pp. 716–722). IEEE. https://doi.org/10.1109/WSC.1994.717419

Chwif, L., Barretto, M. R. P., & Paul, R. J. (2000). On simulation model complexity. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *2000 Winter Simulation Conference Proceedings* (pp. 449–455). IEEE. https://doi.org/10.1109/WSC.2000.899751

Creswell, J. W. (2009). *Research design: Qualitative, quantitative, and mixed methods approaches* (3rd ed.). Sage Publications.

Darema F. (2004) Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In M. Bubak, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), *Computational Science - ICCS 2004 4th International Conference* (pp. 662–669). Springer. https://doi.org/10.1007/978-3-540-24688-6_86

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. In R. Agrawal & P. Stolorz (Eds.), *KDD'98: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 16–22). https://dl.acm.org/doi/10.5555/3000292.3000296

Davis, P. K., & Bigelow, J. H. (1998). *Experiments in multiresolution modeling (MRM).* RAND. https://www.rand.org/content/dam/rand/pubs/monograph_reports/2007/MR1004.pdf

Davis, P. K., & Bigelow, J. H. (2003). *Motivated metamodels: Synthesis of cause-effect reasoning and statistical metamodeling.* RAND. https://www.rand.org/content/dam/rand/pubs/monograph_reports/2005/MR1570.pdf

Davis, P. K., & Tolk, A. (2007). Observations on new developments in composability and multi-resolution modeling. In S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, & R. R. Barton (Eds.), *2007 Winter Simulation Conference Proceedings* (pp. 859–870). https://dl.acm.org/doi/10.5555/1351542.1351697

Dennis, J. B. (2011). Petri Nets. In D. Padua (Ed.), *Encyclopedia of parallel computing* (pp. 1525–1530). Springer. https://doi.org/10.1007/978-0-387-09766-4_134

Department of the Army. (2019). *Tank Platoon (ATP 3-20.15).* Army Techniques Publication. https://armypubs.army.mil/ProductMaps/PubForm/Details.aspx?PUB_ID=1007320

Deslandres, V., & Pierreval, H. (1991). An expert system prototype assisting the statistical validation of simulation models. *SIMULATION, 56*(2), 79–89. https://doi.org/10.1177/003754979105600204

Dodge, Y. (2008). *The conscise encyclopedia of statistics.* Springer.

dos Santos Garcia, C., Meincheim, A., Faria Junior, E. R., Dallagassa, M. R., Sato, D. M. V., Carvalho, D. R., Santos, E. A. P., & Scalabrin, E. E. (2019). Process mining techniques and applications – A systematic mapping study. *Expert Systems with Applications, 133*, 260–295. Elsevier. https://doi.org/10.1016/j.eswa.2019.05.003

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In A. Prieditis & S. Russell (Eds.), *Machine Learning Proceedings 1995* (pp. 194–202). Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-377-6.50032-3

Dunham, M. H. (2002). *Data mining: Introductory and advanced topics.* Prentice Hall.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, & U. Fayyad (Eds.), *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (pp. 226–231). https://dl.acm.org/doi/10.5555/3001460.3001507

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996a). From data mining to knowledge discovery: An overview. *AI Magazine, 17*(3), 37–54. https://doi.org/10.1609/aimag.v17i3.1230

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996b). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM, 39*(11), 27–34. https://doi.org/10.1145/240455.240464

Field, A. (2013). *Discovering Statistics using IBM SPSS Statistics* (5th ed.). Sage Publications.

Filip, F.-G., & Leiviskä, K. (2009). Large-scale complex systems. In S. Y. Nof (Ed.), *Springer Handbook of Automation* (pp. 619–638). Springer. https://doi.org/10.1007/978-3-540-78831-7_36

Fishman, G. S. (1973). *Concepts and methods in discrete event digital simulation.* Wiley.

Fishwick, P. A. (1986). *Hierarchical reasoning: Simulating complex processes over multiple levels of abstraction* [Doctoral dissertation, University of Pennsylvania, US].

Fishwick, P. A. (1988). Role of process abstraction in simulation. *IEEE Transactions on Systems, Man, and Cybernetics, 18*(1), 18–39. https://doi.org/10.1109/21.87052

Fishwick, P. A. (1989). Abstraction level traversal in hierarchical modeling. In B. P. Zeigler , M. Elzas, & T. Oren (Eds.), *Modelling and simulation methodology: Knowledge systems paradigms* (pp. 393–429). Elsevier.

Flood, R. L., & Carson, E. (1993). *Dealing with complexity: An introduction to the theory and application of systems science.* Springer.

Fournier-Viger, P., Lin, J. C.-W., Truong-Chi, T., & Nkambou, R. (2019). A survey of high utility itemset mining. In P. Fournier-Viger, J. C.-W. Lin, R. Nkambou, B. Vo, & V. S. Tseng (Eds.), *High-utility pattern mining: Theory, algorithms and applications* (pp. 1–45). Springer. https://doi.org/10.1007/978-3-030-04921-8_1

Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.-W., & Tseng, V. (2014). SPMF: A java open-source pattern mining library. *Journal of Machine Learning Research. 15*(1). 3389–3393. https://dl.acm.org/doi/10.5555/2627435.2750353

Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Soltani, A., Deng, Z., & Lam, H. T. (2016). The SPMF open-source data mining library version 2. In B. Berendt, B. Bringmann, É. Fromont, G. Garriga, P. Miettinen, N. Tatti, & V. Tresp (Eds.), *Machine Learning and Knowledge Discovery in Databases European Conference ECML PKDD 2016,* (pp. 36-40). Springer. https://doi.org/10.1007/978-3-319-46131-1_8

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., & Koh, Y. S. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition*, *1* (1), 54-77.

Fournier-Viger, P., Yang, Y., Yang, P., Lin, J. C.-W., & Yun, U. (2020). TKE: Mining Top-K frequent episodes. In H. Fujita, P. Fournier-Viger, M. Ali, & J. Sasaki (Eds.), *IEA/AIE 2020: Trends in artifical intelligence theory and applications. Artificial Intelligence Practices. Vol. 12144* (pp. 832–845). Springer. https://doi.org/10.1007/978-3-030-55789-8_71

Frantz, F. K. (1995). A taxonomy of model abstraction techniques. In C. Alexopoulos & K. Kang (Eds.), *1995 Winter Simulation Conference Proceedings* (pp. 1413–1420). https://doi.org/10.1145/224401.224834

Fu, T.C. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence, 24*(1), 164-181. https://doi.org/10.1016/j.engappai.2010.09.007

Fujimoto, R. M. (2001). Parallel and distributed simulation systems. In B. A. Peters, J. S. Smith, D. J. Medeiros, & M. W. Rohrer (Eds.), *2001 Winter Simulation Conference Proceedings* (pp. 147–157). IEEE. https://doi.org/10.1109/WSC.2001.977259

Fujimoto, R. M. (2015). Parallel and distributed simulation. In L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, & M. D. Rossetti (Eds.), *2015 Winter Simulation Conference Proceedings* (pp. 45–59). https://dl.acm.org/doi/10.5555/2888619.2888624

Fujimoto, R. M. (2016). Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation, 26*(4), Article 22. https://doi.org/10.1145/2866577

Gaines, B. R. (1979). General systems research: quo vadis? *General Systems, 24*, 1–9.

Galliers, R. (1991). Choosing appropriate information systems research approaches: A revised taxonomy. In Nissen, H.-E., Klein, H. K. , & Hirschheim, R. (Eds.), *Contemporary approaches and emerging traditions* (pp. 327–345). Elsevier.

Gan, W., Lin, J. C.-W., Fournier-Viger, P., Chao, H.-C., & Yu, P. S. (2019). A survey of parallel sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data, 13*(3), 1–34. https://doi.org/10.1145/3314107

Giambiasi, N., & Frydman, C. (2014). Timed synchronizing sequences. *DEVS'14: Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, Article 21. https://dl.acm.org/doi/10.5555/2665008.2665029

Gionis, A., & Mannila, H. (2003). Finding recurrent sources in sequences. *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology,* 123–130. https://doi.org/10.1145/640075.640091

Goodarzi, M., Freitas, M. P., & Jensen, R. (2009). Feature selection and linear/nonlinear regression methods for the accurate prediction of glycogen synthase kinase-3β inhibitory activities. *Journal of Chemical Information and Modeling, 49*(4), 824–832. https://doi.org/10.1021/ci9000103

Grossmann, W., & Rinderle-Ma, S. (2015). *Fundamentals of Business Intelligence*. Springer. https://doi.org/10.1007/978-3-662-46531-8

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

Hand, D. J., Smyth, P., & Mannila, H. (2001). *Principles of data mining*. MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Springer. https://doi.org/10.1007/978-0-387-84858-7

Henriksen, J. O. (2008). Taming the complexity dragon. *Journal of Simulation, 2*(1), 3–17. https://doi.org/10.1057/palgrave.jos.4250029

Hester, P. T., & Collins, A. (2012). Mathematical applications for combat modeling. In A. Tolk (Ed.), *Engineering principles of combat modeling and distributed simulation* (pp. 385–412). Wiley. https://doi.org/10.1002/9781118180310.ch18

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Hitchins, D. K. (2008). *Systems engineering: A 21st century systems methodology*. John Wiley & Sons. https://doi.org/10.1002/9780470518762

Hoad, K., Robinson, S., & Davies, R. (2010). Automated selection of the number of replications for a discrete-event simulation. *Journal of the Operational Research Society, 61*(11), 1632–1644. https://doi.org/10.1057/jors.2009.121

Hofmann, M. A. (2004). Criteria for decomposing systems into components in modeling and simulation: Lessons learned with military simulations. *SIMULATION, 80*(7-8), 357–365. https://doi.org/10.1177/0037549704049876

Holland, J. H. (1996). *Hidden order: How adaptation builds complexity*. Addison Wesley Longman Publishing.

Holland, J. H. (2000). *Emergence: From chaos to order*. Oxford University Press.

Honig, H. J., & Seck, M. (2012). ΦDEVS: phase based discrete event modeling. *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium,* Article 39. https://dl.acm.org/doi/10.5555/2346616.2346655

Howe, C. (2002). How to Research, 2nd edition by Loraine Blaxter, Christina Hughes & Malcolm Tight, Open University Press, Buckingham, 2001. *Journal of Advanced Nursing, 37*(1), 116–116. https://doi.org/10.1046/j.1365-2648.2002.2089f.x

Hu, X., Zeigler, B. P., & Mittal, S. (2005). Variable structure in DEVS component-based modeling and simulation. *SIMULATION, 81*(2), 91–102. https://doi.org/10.1177/0037549705052227

Huang, K.-Y., & Chang, C.-H. (2008). Efficient mining of frequent episodes from complex sequences. *Information Systems, 33*(1), 96–114. https://doi.org//10.1016/j.is.2007.07.003

Huang, L., Chen, H., Wang, X., & Chen, G. (2000). A fast algorithm for mining association rules. *Journal of Computer Science and Technology, 15*(6), 619–624. https://doi.org/10.1007/BF02948845

Jacobs, P. (2005). The DSOL simulation suite: Enabling multi-formalism simulation in a distributed context [Doctoral dissertation, Delft University of Technology]. http://resolver.tudelft.nl/uuid:4c5586e2-85a8-4e02-9b50-7c6311ed1278

Jacobs, P., & Verbraeck, A. (2006). Mastering D-SOL: A Java based suite for simulation. Delft University of Technology, Faculty of Technology, Policy and Management, Systems Engineering Group, Delft. Retrieved from https://simulation.tudelft.nl/files/tutorial/tutorial_1.6_20060828.pdf

Kailath, T. (1967). The divergence and Bhattacharyya distance measures in signal selection. *IEEE Transactions on Communication Technology, 15*(1), 52–60. https://doi.org/10.1109/TCOM.1967.1089532

Kantardzic, M. (2011). *Data mining: Concepts, models, methods, and algorithms*. John Wiley & Sons.

Keirstead, J., Jennings, M., & Sivakumar, A. (2012). A review of urban energy system models: Approaches, challenges and opportunities. *Renewable and Sustainable Energy Reviews, 16*(6), 3847–3866. https://doi.org/10.1016/j.rser.2012.02.047

Kemper, P., & Tepper, C. (2007). Automated analysis of simulation traces - Separating progress from repetitive behavior. *Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007*), 101-110. https://doi.org/10.1109/QEST.2007.41

Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov Chain. *The Annals of Mathematical Statistics, 24*(3), 338–354. https://doi.org/10.1214/aoms/1177728975

Keogh, E., Chu, S., Hart, D., & Pazzani, M. (2004). Segmenting time series: A survey and novel approach, *Data Mining in Time Series Databases, 57*, 1–21. https://doi.org/10.1142/9789812565402_0001

Kim, B. S., Kang, B. G., Choi, S. H., & Kim, T. G. (2017). Data modeling versus simulation modeling in the big data era: Case study of a greenhouse control system. *SIMULATION, 93*(7), 579–594. https://doi.org/10.1177/0037549717692866

Kiviat, P. J. (1967). *Digital computer simulation: Modeling concepts*. RAND. https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM5378.pdf

Kiviat, P. J. (1969). *Digital computer simulation: Computer programming languages*. RAND. https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM5883.pdf

Kleijnen, J. P. C. (1984). Statistical analysis of steady-state simulations: Survey of recent progress. *European Journal of Operational Research*, *17*(2), 150–162. https://doi.org/10.1016/0377-2217(84)90229-7

Kleijnen, J. P. C. (1987). *Statistical tools for simulation practitioners*. Marcel Dekker.

Kleijnen, J. P. C. (2015). *Design and analysis of simulation experiments* (2nd ed.). Springer. https://doi.org/10.1007/978-3-319-18087-8

Kleijnen, J. P. C. (2017). Design and analysis of simulation experiments: Tutorial. In A. Tolk, J. Fowler, G. Shao, & E. Yücesan (Eds.), *Advances in modeling and simulation: Seminal research from 50 years of Winter Simulation Conferences* (pp. 135–158). Springer. https://doi.org/10.1007/978-3-319-64182-9_8

Klir, G. J. (1985). *Architecture of systems problem solving*. Springer. https://doi.org/10.1007/978-1-4757-1168-4

Klir, G. J. (2001). Systems knowledge. In G. J. Klir (Ed.), *Facets of systems science: Second edition* (pp. 123–134). Springer. https://doi.org/10.1007/978-1-4615-1331-5_7

Klir, G. J., & Elias, D. (2012). *Architecture of systems problem solving*. Springer.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2007). Data preprocessing for supervised leaning. *World Academy of Science, Engineering and Technology, International Journal of Computer and Information Engineering, 1*, 4091–4096. doi.org/10.5281/zenodo.1082415

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics, 22*(1), 79–86. https://doi.org/10.1214/aoms/1177729694

Kwon, Y., Park, H., Jung, S., & Kim, T. (1996). Fuzzy-DEVS formalism: Concepts, realization and applications. *Proceedings of AIS*, 227–234.

L'Ecuyer, P., Meliani, L., & Vaucher, J. (2003). SSJ: A framework for stochastic simulation in Java. In E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes (Eds.), *2002 Winter Simulation Conference Proceedings* (pp. 234–242). IEEE. https://doi.org/10.1109/WSC.2002.1172890

Lane, D. (2006). Hierarchy, complexity, society. In D. Pumain (Ed.), *Hierarchy in natural and social sciences* (pp. 81-119). Springer. https://doi.org/10.1007/1-4020-4127-6_5

Lang, N. A., Jacobs, P. H., & Verbraeck, A. (2003). Distributed open simulation model development with DSOL services. In A. Verbraeck, & V. Hlupic (Eds.), *Simulation in Industry - 15th European Simulation Symposium 2003* (pp. 210–218) SCS-European Publising House.

Lantz, B. (2013). The large sample size fallacy. *Scandinavian Journal of Caring Sciences, 27*(2), 487–492. https://doi.org/10.1111/j.1471-6712.2012.01052.x

Law, A. M. (2015). *Simulation modeling and analysis* (5th ed.). McGraw-Hill.

Laxman, S., & Sastry, P. S. (2006). A survey of temporal data mining. *Sadhana, 31*(2), 173–198. https://doi.org/10.1007/BF02719780

Lee, K., & Fishwick, P. A. (1996). Dynamic model abstraction. In J. M. Charnes, D. M. Morrice, D. T. Brunner, & J. J. Swain (Eds.), *1996 Winter Simulation Conference Proceedings* (pp. 764–771). IEEE. https://doi.org/10.1145/256562.256806

Li, K.H., & Li, C.T. (2019). Linear combination of independent exponential random variables. *Methodology and Computing in Applied Probability*, *21*, 253–277. https://doi.org/10.1007/s11009-018-9653-0

Li R., Liu M., Xu D., Gao J., Wu F., Zhu L. (2022). A Review of Machine Learning Algorithms for Text Classification. In W. Lu, Y. Zhang, W. Wen, H. Yan, & C. Li (Eds.), *Cyber Security. 18th China Annual Conference, CNCERT 2021* (pp. 226–234). Springer. https://doi.org/10.1007/978-981-16-9229-1_14

Lin, M., Lucas, H. C., & Shmueli, G. (2013). Research commentary—Too big to fail: Large samples and the p-value problem. *Information Systems Research*, *24*(4), 906–917. https://doi.org/10.1287/isre.2013.0480

Lin, W., Orgun, M. A., & Williams, G. J. (2002). An overview of temporal data mining. In S. J. Simoff, G. J. Williams, & M. Hegland (Eds.), *Proceedings, Australasian Data Mining Workshop, ADM02* (pp. 83–89). University of Technology, Sydney.

Lin, Y.-F., Huang, C.-F., & Tseng, V. S. (2017). A novel methodology for stock investment using high utility episode mining and genetic algorithm. *Applied Soft Computing, 59*, 303–315. https://doi.org/10.1016/j.asoc.2017.05.032

Liu, D. (2015). *Systems engineering: Design principles and models*. CRC Press. https://doi.org/10.1201/9781315273860

Liu, H., Hussain, F., Tan, C. L., & Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery, 6*(4), 393–423. https://doi.org/10.1023/A:1016304305535

Liu, M., Fang, S., Dong, H., & Xu, C. (2021). Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems, 58,* 346-361. https://doi.org/10.1016/j.jmsy.2020.06.017

Longo, F. (2011). Advances of modeling and simulation in supply chain and industry. *SIMULATION, 87*, 651–656. https://doi.org/10.1177/0037549711418033

Lovric, M., Milanović, M., & Stamenkovic, M. (2014). Algoritmic methods for segmentation of time series: An overview. *Journal of Contemporary Economic and Business Issues*, *1*(1), 31–53. http://hdl.handle.net/10419/147468

Lu X., Tabatabaei S. A., Hoogendoorn M., Reijers H. A. (2019) Trace clustering on very large event data in healthcare using frequent sequence patterns. In T. Hildebrandt, B. van Dongen, M. Röglinger, & J. Mendling (Eds.), *Business Process Management 17th International Conference, BPM 2019* (pp. 198–215). https://doi.org/10.1007/978-3-030-26619-6_14

Lugaresi, G., & Matta, A. (2020). Generation and tuning of discrete event simulation models for manufacturing applications. In K. -H. Bae, B. Feng, S. Lazarova-Molnar, Z. Zheng, T. Roeder, & R. Thiesing (Eds.), *2020 Winter Simulation Conference Proceedings* (pp. 2707–2718). IEEE. https://doi.org/10.1109/WSC48552.2020.9383870

Lugaresi, G., & Matta, A. (2021). Automated Digital Twins generation for manufacturing systems: A case study. *IFAC PapersOnLine, 54,* 749–754. https://doi.org/10.1016/j.ifacol.2021.08.087

Mabroukeh, N. R., & Ezeife, C. I. (2010). A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys, 43*(1), Article 3. https://doi.org/10.1145/1824795.1824798

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 281–297.

Mahajan, P. S., & Ingalls, R. G. (2004). Evaluation of methods used to detect warm-up period in steady state simulation. In R .G. Ingalls, M. D. Rossetti, J. S. Smith, & B. A. Peters (Eds.), *2004 Winter Simulation Conference Proceedings* (pp. 663–671). https://doi.org/10.1109/WSC.2004.1371374

Mamoulis, N. (2009). Temporal data mining. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems,* 2948–2952. https://doi.org/10.1007/978-0-387-39940-9_393

Mannila, H. (1997). Methods and problems in data mining. In F.N. Afrati, P.G. Kolaitis (Eds.), *Proceedings of the 6th International Conference on Database Theory* (pp. 41–55). https://dl.acm.org/doi/10.5555/645502.656095

Mannila, H., Toivonen, H., & Inkeri Verkamo, A. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery, 1*, 259–289. https://doi.org/10.1023/A:1009748302351

Mäntyjärvi, J., Himberg, J., Korpipää, P., & Mannila, H. (2001). Extracting the context of a mobile device user. *IFAC Proceedings Volumes, 34*(16), 387–392. https://doi.org/10.1016/S1474-6670(17)41555-2

Maria, A. (1997). Introduction to modeling and simulation. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *1997 Winter Simulation Conference Proceedings* (pp. 7–13). https://doi.org/10.1145/268437.268440

Markus, K. A. (2007). Philosophical foundations of quantitative research methodology. *Structural Equation Modeling: A Multidisciplinary Journal, 14*(3), 527–533. https://doi.org/10.1080/10705510701303848

Meadows, D. (2009). *Thinking in systems*. Earthscan.

Mitsa, T. (2010). *Temporal data mining*. Chapman & Hall/CRC. https://doi.org/10.1201/9781420089776

Mor, B., Garhwal, S., & Kumar, A. (2020). A systematic review of hidden markov models and their applications. *Archives of Computational Methods in Engineering, 28,* 1429–1448. https://doi.org/10.1007/s11831-020-09422-4

Mörchen, F. (2006a). Algorithms for time series knowledge mining. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 668–673. https://doi.org/10.1145/1150402.1150485

Mörchen, F. (2006b). *Time series knowledge mining* [Doctoral dissertation, Philipps-University Marburg, Marburg, Germany].

Mörchen, F. (2007). Unsupervised pattern mining from symbolic temporal data. *SIGKDD Explorations Newsletter, 9*(1), 41–55. https://doi.org/10.1145/1294301.1294302

Nance, R. E. (1981). The time and state relationships in simulation modeling. *Communications of the ACM, 24*(4), 173–179. https://doi.org/10.1145/358598.358601

Narahari, Y. (1999). Petri nets. *Resonance, 4*(8), 58–69. https://doi.org/10.1007/BF02837068

Negahban, A., & Smith, J. S. (2014). Simulation for manufacturing system design and operation: Literature review and analysis. *Journal of Manufacturing Systems, 33*(2), 241–261. https://doi.org/10.1016/j.jmsy.2013.12.007

Nelson, B. L. (2016). 'Some tactical problems in digital simulation' for the next 10 years. *Journal of Simulation, 10*(1), 2–11. https://doi.org/10.1057/jos.2015.22

Ören, T. I. (1971). *GEST: general system theory implementor (a combined digital simulation language)* [Doctoral dissertation, The University of Arizona]. Retrieved from http://hdl.handle.net/10150/290258

Ören, T. I., & Zeigler, B. P. (2012). System theoretic foundations of modeling and simulation: A historic perspective and the legacy of A Wayne Wymore. *SIMULATION, 88*(9), 1033–1046. https://doi.org/10.1177/0037549712450360

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research, 2*(1), 1–28. https://doi.org/10.1287/isre.2.1.1

Page, E. H., Nicol, D. M., Balci, O., Fujimoto, R. M., Fishwick, P. A., Ecuyer, P. L., & Smith, R. (1999). Panel: strategic directions in simulation research. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, & G. W. Evans (Eds.), *1999 Winter Simulation Conference Proceedings* (pp. 1509–1520). IEEE. https://doi.org/10.1109/WSC.1999.816887

Patnaik, D., Laxman, S., Chandramouli, B., & Ramakrishnan, N. (2012). Efficient episode mining of dynamic event streams. In M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. Webb, & X. Wu (Eds.), *Proceedings of the 12th International Conference on Data Mining* (pp. 605-614). IEEE.https://doi.org/10.1109/ICDM.2012.84

Pearson, K. (1900). X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 50*(302), 157–175. https://doi.org/10.1080/14786440009463897

Pedrycz, W., & Chen, S.-M. (2014). *Information granularity, big data, and computational intelligence.* Springer. https://dl.acm.org/doi/10.5555/2666121

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. (2001). PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In D. C. Young (Ed.), *Proceedings 17th International Conference on Data Engineering* (pp. 215-224). IEEE. https://doi.org/10.1109/ICDE.2001.914830

Perumalla, K. (2006). Parallel and distributed simulation: traditional techniques and recent advances. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, & R. M. Fujimoto (Eds.), *2006 Winter Simulation Conference Proceedings* (pp. 84–95). IEEE. https://doi.org/10.1109/WSC.2006.323041

Petri, C. A., & Reisig, W. (2008). Petri net. *Scholarpedia, 3*, 6477. http://www.scholarpedia.org/article/Petri_net

Petty, M. D., Franceschini, R. W., & Panagos, J. (2012). Multi-resolution combat modeling. In A. Tolk (Ed.), *Engineering principles of combat modeling and distributed simulation* (pp. 607–640). https://doi.org/10.1002/9781118180310.ch25

Plackett, R. L. (1983). Karl Pearson and the Chi-Squared Test. *International Statistical Review/Revue Internationale de Statistique, 51*(1), 59–72. https://doi.org/10.2307/1402731

Poole, T. G., & Szymankiewicz, J. Z. (1977). *Using simulation to solve problems.* McGraw-Hill.

Praehofer, H. (1991). System theoretic formalisms for combined discrete-continuous system simulation. *International Journal of General Systems, 19*(3), 226–240. https://doi.org/10.1080/03081079108935175

Praehofer, H., & Pree, D. (1993). Visual modeling of DEVS-based multiformalism systems based on higraphs. In G. W. Evans, M. Mollaghasemi, E. C. Russel, & W. E. Biles (Eds.), *1993 Winter Simulation Conference Proceedings* (pp. 595–603).

Pritsker, A. A. B. (1979). *Modeling and analysis using Q-GERT networks* (2nd ed.). Wiley. https://doi.org/10.1057/jors.1978.232

Rawashdeh, M. (2020). *Dividing values into equal sized groups*. MATLAB Central File Exchange. Retrieved September 10, 2020 from https://www.mathworks.com/matlabcentral/fileexchange/41266-dividing-values-into-equal-sized-groups

Reichel, K., Bahier, V., Midoux, C., Parisey, N., Jean-Pierre, M., & Stoeckel, S. (2015). Interpretation and approximation tools for big, dense Markov chain transition matrices in population genetics. Algorithms for Molecular Biology *10*, Article 31. https://doi.org/10.1186/s13015-015-0061-5

Robinson, S. (1997). Simulation model verification and validation: increasing the users' confidence. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *1997 Winter Simulation Conference Proceedings* (pp. 53–59). https://doi.org/10.1145/268437.268448

Robinson, S. (2001). Modes of simulation practice in business and the military. In B. A. Peters, J.S. Smith, D.J. Medeiros, & M.W. Rohrer (Eds.), *2001 Winter Simulation Conference Proceedings* (pp. 805–811). IEEE. https://doi.org/10.1109/WSC.2001.977370

Robinson, S. (2008). Conceptual modelling for simulation Part I: Definition and requirements. *Journal of the Operational Research Society, 59*(3), 278–290. https://doi.org/10.1057/palgrave.jors.2602368

Roddick, J. F., Hornsby, K., & Spiliopoulou, M. (2001). An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In J. F. Roddick & K. Hornsby (Eds.), *Temporal, Spatial, and Spatio-Temporal Data Mining First International Workshop, TSDM 2000* (pp. 147–163). Springer. https://doi.org/10.1007/3-540-45244-3_12

Roungas, B., Meijer, S., & Verbraeck, A. (2018). A framework for optimizing simulation model validation & verification. *International Journal On Advances In Systems and Measurements, 11*, 137–152.

Rubner, Y., Tomasi, C., & Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision, 40*(2), 99–121. https://doi.org/10.1023/A:1026543900054

Rui, X., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks, 16*(3), 645–678. https://doi.org/10.1109/TNN.2005.845141

Salt, J. D. (1993). Simulation should be easy and fun! In G. W. Evans, M. Mollaghasemi, E. C. Russel, & W. E. Biles (Eds.), *1993 Winter Simulation Conference Proceedings* (pp. 1–5). IEEE. https://doi.org/10.1145/256563.256567

Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation, 7*(1), 12–24. https://doi.org/10.1057/jos.2012.20

Saysel, A. K., & Barlas, Y. (2006). Model simplification and validation with indirect structure validity tests. *System Dynamics Review, 22*(3), 241–262. https://doi.org/10.1002/sdr.345

Schakel, W. J., Knoop, V. L., & van Arem, B. (2012). Integrated lane change model with relaxation and synchronization. *Transportation Research Record*, *2316*(1), 47–57. https://doi.org/10.3141/2316-06

Schakel, W. J. (2015). *Development, simulation and evaluation of In-car advice on headway, speed and lane* [Doctoral dissertation, Delft University of Technology]. https://doi.org/10.4233/uuid:6dc90efe-6dca-4e0e-81c2-60774e30dd0e

Schlesinger, S., Crosbie, R. E., Gagne, R. E., Innis, G. S., Lalwani, C. S., & Loch, J. (1979). Terminology for model credibility. *SIMULATION, 32*(3), 103–104. https://doi.org/10.1177/003754977903200304

Schriber, T. J. (1989). Perspectives on simulation using GPSS. In C. Alexopoulos, & K. Kang, (Eds.), *1995 Winter Simulation Conference Proceedings* (pp. 451–456). https://doi.org/10.1145/224401.224658

Schruben, L. (1983). Simulation modeling with event graphs. *Communications of the ACM, 26*(11), 957–963. https://doi.org/10.1145/182.358460

Schruben, L., Singh, H., & Tierney, L. (1983). Optimal tests for initialization bias in simulation output. *Operations Research, 31*(6), 1167–1178. https://doi.org/10.1287/opre.31.6.1167

Seber, G. A. F., & Wild, C. J. (2003). *Nonlinear regression*. Wiley.

Seck, M., & Verbraeck, A. (2009). DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. *2009 Summer Computer Simulation Conference Proceedings*, 261–266. https://dl.acm.org/doi/10.5555/2349508.2349543

Seo, C., Zeigler, B. P., & Kim, D. (2018). DEVS markov modeling and simulation: formal definition and implementation. In X. Hu, & F. Barros (Eds.), *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences* (pp. 1–12). https://dl.acm.org/doi/10.5555/3213187.3213188

Serfozo, R. (2009). *Basics of Applied Stochastic Processes*. Springer. https://doi.org/10.1007/978-3-540-89332-5_1

Shani, G., Gunawardana, A., & Meek, C. (2011). Unsupervised hierarchical probabilistic segmentation of discrete events. *Intelligent Data Analysis, 15*(4), 483–501. https://doi.org/10.3233/IDA-2011-0479

Shannon, R. E. (1975). *Systems simulation: The art and science*. Prentice-Hall.

Shannon, R. E. (1976). Simulation modeling and methodology. In H. J. Highland, T. J. Schriber, & R. G. Sargent (Eds.), *Proceedings of the 76 Bicentennial Conference on Winter Simulation* (pp. 9–15). https://dl.acm.org/doi/10.5555/800108.803506

Shannon, R. E. (1998). Introduction to the art and science of simulation. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M. S. Manivannan (Eds.), *1998 Winter Simulation Conference Proceedings* (pp. 7–14). https://dl.acm.org/doi/10.5555/293172.293175

Shortle, J., Thompson, J., Gross, D., & Harris, C. (2018). *Fundamentals of Queueing Theory* (5th ed.). Wiley. https://doi.org/10.1002/9781119453765

Šiljak, D. D. (1978). *Large-scale dynamic systems: stability and structure*. North-Holland.

Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society, 106*(6), 467–482.

Simon, H. A. (1991). The architecture of complexity. In G. J. Klir (Ed.), *Facets of Systems Science* (pp. 457–476). Springer. https://doi.org/10.1007/978-1-4899-0718-9_31

Simpson, T., Poplinski, J. D., Koch, P., & Allen, J. (2001). Metamodels for computer-based engineering design: Survey and recommendations. *Engineering with Computers, 17*, 129–150. https://doi.org/10.1007/PL00007198

Siskind, J. M. (1999). Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research., 15*(1), 31–90. https://doi.org/10.1613/jair.790

Skyttner, L. (2006). *General systems theory: Problems, perspectives, practice* (2nd ed.). World Scientific. https://doi.org/10.1142/5871

Sola, A., Corchero, C., Salom, J., & Sanmarti, M. (2020). Multi-domain urban-scale energy modelling tools: A review. *Sustainable Cities and Society, 54*, 101872. https://doi.org/10.1016/j.scs.2019.101872

Song, M., Günther, C. W., & van der Aalst, W. M. P. (2009). Trace clustering in process mining. In D. Ardagna, M. Mecella, & J. Yang (Eds.), *Business Process Management Workshops, BPM 2008 International Workshops* (pp. 109–120). Springer. https://doi.org/10.1007/978-3-642-00328-8_11

Sorensen, D., & Gianola, D. (2007). *Likelihood, Bayesian, and MCMC methods in quantitative genetics.* Springer. https://doi.org/10.1007/b98952

Sulistio, A., Yeo, C. S., & Buyya, R. (2004). A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software: Practice and Experience, 34*(7), 653–673. https://doi.org/10.1002/spe.585

Sullivan, G. M., & Feinn, R. (2012). Using effect size—or why the p value is not enough. *Journal of Graduate Medical Education, 4*(3), 279–282. https://doi.org/10.4300/jgme-d-12-00156.1

Tamminga, G. (2019). *A novel design of the transport infrastructure for traffic simulation models* [Doctoral dissertation, Delft University of Technology]. https://doi.org/10.4233/uuid:35d2e152-0cfe-439e-a276-da4a69b11acd

Taylor, S. J. E., Khan, A., Morse, K. L., Tolk, A., Yilmaz, L., Zander, J., & Mosterman, P. J. (2015). Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens. *SIMULATION*, *91*(7), 648–665. https://doi.org/10.1177/0037549715590594

Tekinay, C., Seck, M. D., Fumarola, M., & Verbraeck, A. (2010). A context-based multi-perspective modeling and simulation framework. In B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, & E. Yücesan (Eds.), *2010 Winter Simulation Conference Proceedings* (pp. 479–489). IEEE. https://doi.org/10.1109/WSC.2010.5679137

Tekinay, C., Seck, M. D., & Verbraeck, A. (2012). Exploring multi-level model dynamics: Performance and accuracy (WIP). *2012 Symposium on Theory of Modeling and Simulation Proceedings,* Article 20. https://dl.acm.org/doi/10.5555/2346616.2346636

Tolk, A. (2012). *Engineering principles of combat modeling and distributed simulation*. John Wiley & Sons.

Tolk, A. (2015). The next generation of modeling & simulation: integrating big data and deep learning. In S. Mittal, I-C. Moon, & E. Syriani (Eds.), *2015 Summer Computer Simulation Conference Proceedings* (pp. 1–8). https://dl.acm.org/doi/10.5555/2874916.2874964

Truong-Chi, T., & Fournier-Viger, P. (2019). A survey of high utility sequential pattern mining. In P. Fournier-Viger, J. C.-W. Lin, R. Nkambou, B. Vo, & V. S. Tseng (Eds.), *High-utility pattern mining: Theory, algorithms and applications* (pp. 97–129). Springer. https://doi.org/10.1007/978-3-030-04921-8_4

van der Aalst, W. M. P. (2011). Process mining: Discovery, conformance and enhancement of business processes. Springer. https://doi.org/10.1007/978-3-642-19345-3

van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action*. Springer. https://doi.org/10.1007/978-3-662-49851-4_1

van Lint, H., Calvert, S., Schakel, W., Wang, M., & Verbraeck, A. (2018). Exploring the effects of perception errors and anticipation strategies on traffic accidents - A simulation study. In D. N. Cassenti (Ed.), *Advances in human factors in simulation and modeling* (pp. 249–261). Springer. https://doi.org/10.1007/978-3-319-60591-3_23

van Lint, H., Schakel, W., Tamminga, G., Knoppers, P., & Verbraeck, A. (2016). Getting the human factor into traffic flow models: New open-source design to simulate next generation of traffic operations. *Transportation Research Record, 2561*(1), 25–33. https://doi.org/10.3141/2561-04

Vangheluwe, H. L. M. (2008). Foundations of modelling and simulation of complex systems. In C. Ermel, R. Heckel, J. de Lara, T. Margaria, J. Padberg, & G. Taentzer (Eds.), *Proceedings of the Seventh International Workshop on Graph Transformation and Visual Modeling Techniques* (pp. 1–12). https://doi.org/10.14279/tuj.eceasst.10.162

Vangheluwe, H. L. M., & de Lara, J. (2002). Meta-models are models too. In E. Yücesan, C.-H. Chen, J. L. Snowdon, & J. M. Chames (Eds.), *2002 Winter Simulation Conference Proceedings* (pp. 597–605). IEEE. https://doi.org/10.1109/WSC.2002.1172936

Vangheluwe, H. L. M. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. *IEEE International Symposium on Computer-Aided Control System Design Conference Proceedings* (pp. 129–134). IEEE. https://doi.org/10.1109/CACSD.2000.900199

Villafane, R., Hua, K. A., Tran, D., & Maulik, B. (1999). Mining interval time series. In M. Mohania & A. Min Tjoa (Eds.), *DataWarehousing and Knowledge Discovery First International Conference* (pp. 318–330). Springer. https://doi.org/10.1007/3-540-48298-9_34

von Bertalanffy, L. (1968). *General System Theory: Foundations, development, applications*. G. Braziller.

Wainer, G. (2002). CD++: a toolkit to develop DEVS models. *Software: Practice and Experience, 32*(13), 1261–1306. https://doi.org/10.1002/spe.482

Wainer G., & Giambiasi N. (2001) Timed Cell-DEVS: Modeling and simulation of cell spaces. In H. S. Sarjoughian & F. E. Cellier (Eds.), *Discrete event modeling and simulation technologies* (pp. 187–214). Springer. https://doi.org/10.1007/978-1-4757-3554-3_10

Wainer, G. A., & Giambiasi, N. (2002). N-dimensional Cell-DEVS Models. *Discrete Event Dynamic Systems, 12*(2), 135–157. https://doi.org/10.1023/a:1014536803451

Wang, Y., Li, J., Hongbo, S., Li, Y., Akhtar, F., & Imran, A. (2019). A survey on VV&A of large-scale simulations. *International Journal of Crowd Science*, *3*(1), 63–86. https://doi.org/10.1108/IJCS-01-2019-0004

Weinberg, G. (1975). *An introduction to general systems thinking*. Wiley.

Whitt, W. (1991). The efficiency of one long run versus independent replications in steady-state simulation. *Management Science, 37*(6), 645–666. https://doi.org/10.1287/mnsc.37.6.645

Wieland, F., & Pritchett, A. (2007). Looking into the future of air transportation modeling and simulation: A grand challenge. *SIMULATION, 83*(5), 373–384. https://doi.org/10.1177/0037549707078851

Witten, I. H., & Frank, E. (2002). Data mining: practical machine learning tools and techniques with Java implementations. *ACM Special Interest Group on Management of Data (SIGMOD) Record, 31*(1), 76–77. https://doi.org/10.1145/507338.507355

Wong, W., Wong, S. C., & Liu, H. X. (2021) Network topological effects on the macroscopic fundamental diagram. *Transportmetrica B: Transport Dynamics*, *9*(1), 376-398, https://doi.org/10.1080/21680566.2020.1865850

Wu, C.-W., Lin, Y.-F., Yu, P. S., & Tseng, V. S. (2013). Mining high utility episodes in complex event sequences. In R. Ghani, T.E. Senator, P. Bradley, R. Parekh, & J. He (Eds.), *2013 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Proceedings* (pp. 536–544). https://doi.org/10.1145/2487575.2487654

Wymore, W. (1967). *A mathematical theory of systems engineering: The elements*. Wiley.

Yan, X., & Su, X. G. (2009). *Linear regression analysis: Theory and computing*. World Scientific Publishing. https://doi.org/10.1142/6986

Yilmaz, L., & Oren, T. I. (2004). Dynamic model updating in simulation with multimodels: A taxonomy and a generic agent-based architecture. In A. G. Bruzzone & E. Williams (Eds.), *2004 Summer Computer Simulation Conference Proceedings* (pp. 3–8). https://site.uottawa.ca/~oren/pubs/pubs-2004-04-SCSC-MM.pdf

Yilmaz, L., & Tolk, A. (2006). Engineering ab initio dynamic interoperability and composability via agent-mediated introspective simulation. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, & R. M. Fujimoto (Eds.), *2006 Winter Simulation Conference Proceedings* (pp. 1075–1082). IEEE. https://doi.org/10.1109/wsc.2006.323197

Yilmaz, L., Lim, A., Bowen, S., & Oren, T. (2007). Requirements and design principles for multisimulation with multiresolution, multistage multimodels. In S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, & R. R. Barton (Eds.), *2007 Winter Simulation Conference Proceedings* (pp. 823–832). IEEE. https://doi.org/10.1109/WSC.2007.4419678

Zeigler, B. P. (2019). Chapter 2 - Simulation-based evaluation of morphisms for model library organization. In L. Zhang, B. P. Zeigler, & Y. laili (Eds.), *Model engineering for simulation* (pp. 25–42). Academic Press. https://doi.org/10.1016/B978-0-12-813543-3.00002-0

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems* (2nd ed.). Academic Press.

Zeigler, B. P., Muzy, A., & Kofman, E. (2019). Chapter 21 - DEVS Markov Modeling and Simulation. In B. P. Zeigler, A. Muzy, & E. Kofman (Eds.), *Theory of modeling and simulation* (3rd ed.) (pp. 567–599). Academic Press. https://doi.org/10.1016/B978-0-12-813370-5.00032-8

Zeigler, B. P., & Praehofer, H. (1989). Systems theory challenges in the simulation of variable structure and intelligent systems. In F. Pichler & R. Moreno-Diaz (Eds.), *Computer Aided Systems Theory — EUROCAST '89* (pp. 41–51). Springer. https://doi.org/10.1007/3-540-52215-8_4

Zhang, X. (2018). Application of discrete event simulation in health care: a systematic review. *BMC Health Services Research, 18***,** Article 687. https://doi.org/10.1186/s12913-018-3456-4

Zhao, Q., & Bhowmick, S. S. (2003). *Sequential pattern mining: A survey. Technical Report No. 2003118*. CAIS, Nanyang Technological University, Singapore. Retrieved from https://cs.nju.edu.cn/zhouzh/zhouzh.files/course/dm/reading/reading04/zhao_techrep03.pdf

Acknowledgements

Acknowledgements

It was Friday morning, August 28, 2009. I was sitting in my window seat, anxiously waiting for my plane to take off. Part of me was excited about this trip, as it was the beginning of an entirely new chapter for me. A dream scenario, if you will. I was going to do a Ph.D. at TU Delft. Yet, I could not stop myself from feeling very nervous. The thought of my first plane flight ever, taking me on my first trip abroad ever, where I would be living on my own without friends and family for the first time ever, was racing my heart. I reached out to my pocket, took out my headphones, and started playing the song that I had on repeat for the past few days: *Asfalt Dünya - Sakın*. Then, I took a deep breath and told myself that it would be alright. I would finish my Ph.D. in four years, come back to Turkey, and continue from where I left. Little did I know that four years would become twelve years, and that the Netherlands would become more than a temporary destination. Although this "longer than planned" journey felt challenging many times, it also brought me irreplaceable memories, inspiring collaborations, and amazing friends. I cannot express how grateful I am to all of those I was lucky to meet along the way.

First and foremost, I would like to thank my promotor, **Prof. Alexander Verbraeck**, for his invaluable guidance, continuous support, and patience during the past years. Alexander, your enthusiasm towards complex research challenges and your endless confidence in me are the reasons I was able to push through and complete this dissertation after all these years. Moreover, your high standards and expectations, but also the fact that you allowed me to make and learn from my mistakes, helped me grow immensely both on a personal and professional level and deliver a dissertation beyond what I could imagine when I started. Thank you for believing in me, even when I did not believe in myself.

I would also like to thank **Dr. Mamadou Seck** for his guidance and friendship during his time at the Systems Engineering Department as my daily supervisor. Mamadou, thank you for the inspiring discussions on potential research directions. The advice and guidance you gave me at the early stages of this research were invaluable.

I would like to thank **Prof. Hans Vangheluwe**, **Prof. Simon Taylor**, **Prof. Martijn Warnier**, **Prof. Jan Kwakkel**, **Dr. Yilin Huang**, and **Dr. Gönenç Yücel** for accepting to be a part of my doctoral committee and reading the draft dissertation.

Many thanks to my former colleagues and friends that were present during my time at the Systems Engineering section. **Michele Fumarola** and **Yilin Huang**, thank you for the discussions and collaborations on simulation related research. Together with **Evangelos Pournaras**, **Jordan Janeiro**, **Kassidy Clark**, **Jan-Paul van Staalduinen**, and **Shalini Kurapati**, you made sure that I had plenty of opportunities to relax after work hours. I will always remember our time together with a huge smile on my face. I would like to thank **Michel Oey**, **Martijn Warnier**, **Joseph Barjis**, **Rens Kortmann**, **Yakup Koç**, **Deniz Çetinkaya** and other colleagues of the Systems Engineering section for discussing and helping me with aspects of my work on multiple occasions. Thank you all for being such great colleagues and friends.

A special word for my dear paranymphs **Çiğdem** & **Burak**. Thank you for standing beside me on the day of my defense, just like you have been standing by my side since the start of this journey. I find it difficult to find the right words to express how deeply I care about you both. I want you to know that you are like a sister and a brother to me and I feel blessed to have you in my life.

To the *Şamandıring* family and the wonderful people I met through you: **Argun**, **Berk**, **Burak** (& **Rose**), **Çiğdem** (& **Thomas**), **Esin**, **Esra**, **Feyza**, **Güncem**, **Nazlı** (& **Martijn**), **Onur**, **Sine** (& **Daniel**), and **Taner**. Thank you all for brightening up my life with your amazing friendship, and my phone screen with your messages. Our trips and parties are some of my best memories. Although I have not been much available in the past two years, I will ensure that you will have enough of me for the rest of your lives. I cannot wait to make new memories with you all. **Argun**, thank you for being such a wonderful roomie during my time at Bosboom. I will always cherish the fun times we had together. **Taner**, you have literally been with me on this journey from day one. You have been a big brother teaching me how to cook pasta, a friend showing me how to enjoy the perks of life, and a mentor guiding me in my professional growth. Thank you for being the great person you are.

To the other amazing people that I have met over the years in the Netherlands: **Sinan**, **Noyan**, **Vanya**, **Onursal**, **Mert**, **Mülazım**, **Caner**, **Nalan**, **Naim Kenan**, **Özgür** (& **Tessa**), **Gönenç**, **İlhan**, **Elvin** (& **Jaap**) and many others whom I am not able to individually mention here. Without you, my time in the Netherlands would not have been as great. Thank you all for everything. **Sinan**, thank you for pushing me out of my comfort zone and getting me back on two wheels. I will never forget our Amsterdam and Eindhoven Gran Fondo rides, as well as the two Limburgs Mooiste together with **Argun**. Those are some of my greatest personal achievements. I am also confident that my lower back will forever remind me of the first Limburgs Mooiste ride.

I would like to extend my thanks and appreciation to all my former colleagues and friends at *Quintiq*. Thank you all for your friendships that made the office a welcoming place for me. Special thanks to **Lindsay**, **Dirk**, **Jelle**, **Arthur**, **Thomas**, and **Diederik** for being such generous colleagues and great friends. I really enjoyed all our conversations, whether it was at the Den Bosch office or in London, Kuala Lumpur, Copenhagen, or Dubai during a business trip. **Dirk** and **Jelle**, I will never forget our adventures in London. Together with **Arthur**, you never failed to turn a regular evening into something memorable, and I am looking forward to having more of those with you. **Lindsay**, working with you was never dull. I will miss all of our amusing conversations about totally random topics, as well as your expert opinions on food, drinks, and travel.

To my dearest *Oni* family*,* **Ayla**, **Barış**, **Burak**, **Ergin**, **Fatih**, **Gazi Çağrı**, **Handan**, **Hüsmen**, **Ilgaz**, **Murat**, **Oktay**, **Salim**, **Semiha**, **Tahsin**, **Türker**, and others whom I am not able to individually thank here. You have been nothing but incredible friends for more than two decades. I have always felt your sincere love and support at every step of my life, starting from the time we were clueless teenagers in school uniforms up until now. You have always been, and you will always be, very special to me, and I will always put you up on a pedestal where you belong. Thank you for everything. **Tahsin**, your memories will stay with me forever. Special thanks to the *Tenezzül*'s core members, **Barış**, **Ergin**, **Gazi Çağrı**, **Hüsmen**, **Oktay**, and **Türker** for making me laugh every day for the last two decades with their messages.

Uğur Ceylan, I truly appreciate you always making time for me when I am in Ankara. You never seem bothered if we haven't seen or spoken to each other in a long time and

always ensure we pick up from where we left. I hope to be a part of your life for many years to come.

Muğla tayfası, **Ilgaz** & **Hülya**, **Ahmet** & **Sema**, and **Samet** & **Derya**. Thank you for making my trips to Turkey something I eagerly anticipate. **Ilgaz** & **Hülya**, my sincere thanks to you for making me feel like a member of your beautiful family, for spoiling me with delicious food and endless hospitality, and for generously sharing your heartfelt laughter and big hearts. I adore you and your lovely family.

To my family in the Netherlands, **Gerrit**, **Sara**, **Elmer** & **Marjolein**, and **Ruben** & **Sanne**. You made me feel welcomed and loved from the first time we met. **Gerrit** and **Sara**, I could not wish for better parents-in-law than you. You are amazing human beings and your love radiates. **Elmer** & **Marjolein,** I have never felt so touched seeing my name on a birthday calendar. It is a joy to be a bonus uncle to your beautiful daughters. **Ruben** & **Sanne**, it is impossible not to admire how kind and generous you both are. Please never change.

Suzanna, *lief*… Your love and support have been my anchor ever since you entered my life, keeping my feet on the ground and my mind still when the waves of self-doubt came crashing down. Your beautiful soul and intelligence have served as my lighthouse, guiding me through the fog of social pressure and self-expectations, revealing to me the true beauty in things. I am a massive fan of your smile welcoming me every morning, your endless and unconditional love, your curiosity and enthusiasm for cultures and travel, your incredible passion for food (both making and eating), your honesty and modesty, your determination and resilience, and your kindness. I would not have completed this dissertation without your support and your patience. For that, I will be forever grateful. The past six years together have been nothing short of amazing, filled with incredible moments and ever-growing love. I cannot wait to make more beautiful memories together and grow our love-moestuin even larger.

My beautiful family, this Ph.D. and everything else in my life are possible thanks to you. Thank you for making me know and feel that I am truly and deeply loved and supported no matter what. Thank you for lighting my way at night and for making sure I can always find my way *home*. **Çağatay**, en yakın dostum, ilk ve tek göz ağrım, sırdaşım

ve yoldaşım, tanıdığım en komik ve koca yürekli ve hayattaki her şeyden çok daha değerli kardeşim. Her ne kadar doktoramı yaptığım süre esnasında yakınımda olamasan, her bitkin ve yenilmiş hissettiğimde sana gelip teselli bulmak için sarılamamış ya da her mutlu anımda yanına koşup seninle paylaşamamış ve birlikte havalara sıçrayıp kutlayamamış olsam da varlığını, desteğini ve en önemlisi hiç tükenmeyen sevgini ne kadar uzaktan olursa olsun her zaman hissettirdiğin için teşekkür ederim. Canım annem ve babam, **Zahide** ve **Kâtip**, her ne kadar uzakta olursam olayım bana daima sevildiğimi ve güvende olduğumu hissettirdiğiniz için sizlere çok teşekkür ederim. Sizlerin fedakârlıkları, tükenmeyen sabrı, sınırsız sevgisi ve bitmeyen güveni olmadan bu doktorayı bitirmem ve bu satırları yazmam mümkün olamazdı. Sizleri çok seviyorum!

Çağrı Tekinay

Delft, June 2022

About the Author

Çağrı Tekinay was born in 1984 in Malatya, Turkey. After graduating from Ankara Ataturk Anatolian High School in 2002, he continued his higher education at Baskent University, where he obtained his B.Sc. degree in Computer Engineering (2007). During his studies, he co-founded the internationally recognized Baskent University Robotics Society. In 2009, he obtained his M.Sc. degree in Information Systems from the Graduate School of Informatics at Middle East Technical University.

After completing his studies, Çağrı started a Ph.D. project under the supervision of Prof. dr. ir. Alexander Verbraeck and Dr. Mamadou D. Seck in the Systems Engineering Section of the Technology, Policy and Management Faculty at Delft University of Technology, the Netherlands. In his research, he introduced a novel approach that combines the fields of modeling & simulation and temporal data mining to automate the abstraction of discrete-event simulation models using state-trace data. During his time in Delft, he served as president (2012) and treasurer (2011) for the Society for Modeling & Simulation (SCS) Student Chapters.

Between 2014 and 2019, Çağrı worked as a Lead Solutions Consultant at Dassault Systèmes/DELMIA Quintiq. In this position, he coordinated teams of consultants and software developers, collaborated with subject matter experts, led on-site and online analysis workshops, and designed and implemented planning and optimization software solutions for complex supply chain puzzles from various industries, including transportation and logistics (e.g., Transport for London), emergency services (e.g., Falck UK & Denmark), and aviation (e.g., Dubai Airports). For instance, he led the software solution design and documentation for Dubai Airports, and coordinated international teams throughout its implementation and delivery phases. As of April 2018, Dubai International Airport (DXB) uses their "Airport Operations and Ground Resources Planning and Optimization Solution" to serve more than 90 million passengers each year. Çağrı has obtained senior-level certificates for both technical and business solutions consultancy and currently works as an independent solutions consultant.

Large-scale complex systems are characterized by a large number of interconnected variables and a diverse set of interactions. As the demand for the development and optimization of large-scale systems is growing, so does the need for better techniques to understand their underlying dynamic behavior and predict and manage their long-term performance. With the increased capabilities of computer technology, we have been able to run simulation models for these systems that are larger in scale and higher in complexity. While these advancements have enabled more accurate representations of real-world systems, the ever-increasing scale and complexity of simulation models may eventually result in models that are too complex to work with – giving rise to *large-scale complex simulation models*.

In this dissertation, we aim to investigate to what extent the abstraction of large-scale complex simulation models, specifically discrete- event simulation models expressed in the DEVS formalism, can be automated using their state- trace data. In order to achieve this objective, we designed a method that integrates the fields of modeling and simulation and temporal data mining by utilizing state-trace data and applying frequent episode mining techniques to discover behavioral patterns.