Reusing Simulation Experiments for Model Composition and Extension

Dissertation

zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von

Danhua Peng, geb. am 26. 05. 1988 in Hengyang, China

aus Rostock

Rostock, 06. 01. 2017

urn:nbn:de:gbv:28-diss2017-0028-8

Principal Advisor:

Prof. Dr. Adelinde M. Uhrmacher,
Institut für Informatik, Universität Rostock

External Reviewers:

1. Prof. Dr. Jane Hillston,
School of Informatics, University of Edinburgh

2. Prof. Dr. François Fages,
INRIA Saclay Île-de-France Research Center

Date of Submission: January 06, 2017
Date of Defense: February 20, 2017

Abstract

As the number of available models as well as their complexity grows, model reuse becomes increasingly interesting. However, building new models based on reusing existing ones poses several challenges, and among others, checking validity of the newly built model and the model reuse is of paramount importance. The modeling process is typically accompanied by executing simulation experiments, e.g., for model validation. Therefore, this thesis aims to employ simulation experiments to support developing models via model reuse, with a focus on validating the resulting model.

The essential idea presented by this thesis is to reuse simulation experiments conducted for model validation. Individual models are annotated with their simulation experiments that have been performed for validating the model. Upon reuse of those models for building new ones, the associated simulation experiments are also reused and executed with the new model, to inspect whether the key behavior exhibited by the original models is preserved or not in the new model. Hence, the changes of model behavior resulting from the model reuse are revealed, and insights into validity of the new model are provided.

As a prerequisite, an explicit, declarative experiment specification containing four aspects is proposed, i.e., model configuration, simulation configuration, data processing method, and model behavioral property. The embedded domain specific language SESSL is exploited and extended for specifying experiments. The tool TAECS is developed to automatically adapt and execute experiments as well as analyzing the results. To allow an automated evaluation of experiment results, simulation-based model checking techniques are adopted and integrated.

Two types of model reuse are considered, i.e., model composition and model extension. Reusing simulation experiments in supporting model development via composition and extension are investigated individually. The effectiveness of the developed approach for model composition is demonstrated by two case studies, a case study of Lotka-Volterra models and a case study based on replaying a realistic simulation study of Wnt/β-catenin pathway. Moreover, to illustrate the benefits of the developed approach for model extension, a case study of receptor ligand pathway is presented.

Zusammenfassung

Mit der zunehmenden Anzahl und Komplexität verfügbarer Modelle wird die Wiederverwendung von Modellen immer interessanter. Das Erstellen neuer Modelle durch die Wiederverwendung bereits vorhandener beinhaltet jedoch mehrere Herausforderungen, wobei unter anderem die Validitätsprüfung des neuen Modells sowie die Modellwiederverwendung von größter Bedeutung sind. Der Modellierungsprozess geht typischerweise mit Simulationsexperimenten einher, wie etwa zur Modellvalidierung. Daher beschäftigt sich diese Arbeit mit der Verwendung von Simulationsexperimenten zur Unterstützung der Modellentwicklung durch Wiederverwendung von Modellen. Der Fokus dieser Arbeit liegt dabei auf der Validierung der entwickelten Modelle.

Die grundlegende Idee dieser Arbeit ist, Validierungsexperimente wiederzuverwenden. Einzelne Modelle sind mit den Simulationsexperimenten annotiert, mit denen sie validiert wurden. Werden diese Modelle wiederverwendet, werden die entsprechenden Simulationsexperimente ebenfalls wiederverwendet und auf das neue Modell angewandt, um festzustellen, ob das zentrale Verhalten der Ausgangsmodelle in dem neuen Modell erhalten geblieben ist. Somit wird geprüft, ob durch die Wiederverwendung des Modells Änderungen im Modellverhalten hervorgerufen wurden, und Erkenntnisse über die Validität des neuen Modells werden ermöglicht.

Als Ausgangspunkt wird eine explizite, deklarative Experimentspezifikation eingeführt, welche die vier Teilaspekte Modelkonfiguration, Simulationskonfiguration, Datenverarbeitungsmethode und Modellverhalten enthält. Um Experimente zu spezifizieren, wird die interne domänenspezifische Sprache SESSL verwendet und erweitert. Das Werkzeug TAECS wurde entwickelt, um automatisch Experimente zu adaptieren und auszuführen sowie deren Ergebnisse zu analysieren. Um eine automatische Auswertung von Experimentergebnissen zu ermöglichen, werden simulationsbasierte Model-Checking Verfahren genutzt und integriert.

Zwei Arten der Modellwiederverwendung werden betrachtet, Modellkomposition und Modellerweiterung. Die Wiederverwendung von Simulationsexperimenten als Unterstützung zur Modellentwicklung wird für beide Fälle im einzelnen untersucht. Die Effektivität des entwickelten Ansatzes zur Modellkomposition wird an zwei Fallstudien veranschaulicht: eine Fallstudie von Lotka-Volterra Modellen und eine Fallstudie basierend auf der Wiederholung einer realistischen Simulationsstudie von Wnt/β-Catenin Signalwegen. Um die Vorteile des entwickelten Ansatzes zur Modellerweiterung aufzuzeigen, wird eine Fallstudie von Rezeptor-Ligand-Signalwegen vorgestellt.

Acknowledgements

Yet another journey is coming to the end, which began four years ago when I firstly arrived in Rostock without knowing anyone in this foreign city. During the last four years, many people have supported me. I offer my deepest gratitude to my supervisor Professor Lin Uhrmacher, for giving me the opportunity to do my PhD in modeling and simulation group at University of Rostock in the first place, and then guiding and supporting me throughout this journey with her knowledge, patience and encouragement. I would thank Professor François Fages and Professor Jane Hillston, who agreed to be the reviewer of this thesis shortly before the submission, especially given that the schedule for reviewing is rather tight.

I would further thank all my current and former colleagues of the modeling and simulation group for the creative and friendly atmosphere, and it was great pleasure to work with you. I thank Roland Ewald, from who I can always learn so much, for helping me in dealing with SESSL and giving valuable advice in building the first prototype of this work. I also thank Tom Warnke, for implementing all algorithms for checking properties in this thesis, helping me with model checking problems, providing helpful suggestions in our discussions, and also translating the abstract of this thesis into German. I am thankful to Fiete Haack, who provided assistance in the modeling part for the last two case studies of this work, and answered my all kinds of questions regarding the models with great patience. Special thanks go to Tobias Helms for always being there to help me out and for being not only a great colleague but also a good friend. I would also like to thank Jan Himmelspach, who helped me to come to Germany for my PhD study.

I am grateful to all my wonderful friends, those I met in Rostock who made my staying here full of joy, and those in China who are always there for me no matter where I am. Also, I would like to acknowledge the China Scholarship Council for the financial support during the four years.

Last but not least, I deeply thank my family for all the support and understanding over those years.

Contents

Chapter 1

Introduction

1.1. Context and Background

Model Development is an Intricate Process

Modeling and simulation (M&S), as a tool and a discipline, has been widely used for system analysis and decision making in many areas such as science, engineering and military. When experimentation on the system itself is not feasible for different reasons, e.g., it is costly, dangerous, time-consuming or the system is not available [30, p. 10], building a model and performing simulation experiments with it can provide knowledge about the system being modeled and answer "what-if" questions. Whether it is viewed as a science or as an art [216], modeling is the most difficult and important part of a simulation study [151].

Model development is an intricate process in which the modeler needs an intimate relationship to the system and phenomena to be modeled, as abstraction and selective enrichment of models are key issues [224]. Undoubtedly, it requires great effort to obtain a satisfactory model. The situation is aggravated in developing models for large complex systems, such as the modeling of brain and climate, which is believed to pose grand challenges [240]. Therefore, a suitable support for the model development process is required to reduce the cost, accelerate the process, and enhance the quality.

The modeling and simulation community has demonstrated a longstanding focus on providing support for the modeling task. A large body of work has been devoted to advancing the development of models from different aspects. To facilitate describing diverse complex properties of the system to be modeled, numerous sophisticated modeling formalisms have been introduced. For example, in the field of systems biology, various types of modeling languages have been proposed to meet different requirements for representing biological systems, such as equation-based formalisms,

rule-based formalisms and constrain-based formalisms [145, 146]. Meanwhile, to accelerate the execution of models, a lot of work exists on how to improve the performance of simulation execution, e.g., efficient simulation algorithms. Furthermore, as no simulation algorithm presents the best performance in all situations, some mechanisms are developed for automatic algorithm selection, e.g., [68]. With the improved data availability, some approaches have been proposed for automated/semi-automated model generation from existing data, such as defining generic models or model templates [143, 258].

As the process of building models is highly complex, a plenty of effort has been dedicated to structure it into different phases, i.e., the modeling and simulation life cycle. Different modeling and simulation life cycles exist, depending on the focus of study and the application, e.g., those presented by Sargent [214] and by Balci [9]. For each identified phase in those life cycles, e.g., verification and validation of simulation models, a plethora of approaches have been developed. Furthermore, some effort has been dedicated to the management of the modeling process, such as integrating workflow systems into this process to provide user guidance and increase reproducibility, e.g., in [206].

While all those existing approaches have achieved progress from different aspects in supporting the modeling process, model development remains an intricate process with many challenges.

Developing Models via Reuse

On the one hand, developing models is non-trivial, which may require a great deal of time, cost, and effort [199]; on the other hand, more and more models are developed and become available. Consequently, building new models by reusing already developed ones comes into play, i.e., *model reuse*, which represents an appealing intuition of reducing the required time and cost in modeling [147, 175].

Model reuse has attracted a large amount of discussions, to name but a few, as presented in [15, 175, 178, 181, 190]. The primary benefits expected from model reuse include the improvement in model development(i.e., reduced developing time and cost), higher quality and more cost effective simulation studies [106], and flexibility of creating reusable objects at different granular levels [201].

Model reuse can refer to different levels, from the reuse of small pieces of code, through the reuse of model components, up to the reuse of complete models [201]. As more and more complete models are developed, their reuse becomes more of interest. Complete models can be reused without modification, for the same purpose or in a different environment [201]. Besides, a particularly interesting usage of reusing

complete models is to build new models, either by composing them with other models, i.e., *model composition*, or by extending them, i.e., *model extension*. For example, to understand the complex phenotypes resulting from individual molecules and their interactions in computational biology, a "whole-cell" model was built by integrating 28 independently developed sub-models, based on over 900 publications [105]. In this case, several complete models were reused to build a large, comprehensive model via model composition so that the enormous complexity of modeling a single cell is divided and becomes manageable. In the analysis of rarefied gas flows, successive model extensions have been applied in the modeling of gas-surface interaction, i.e., the essential Cercignani-Lampis gas-surface scattering model (the C-L model) [31] was extended to include important uncovered cases in [138], the resulting model of which was further extended in [139] to allow capturing more phenomena.

Simulation Experiments in Model Development

Once a satisfactory model is built, simulation experiments are undertaken to fulfill the goals that the model is built for, e.g., to gain understanding of the system being modeled and to provide results of different scenarios for decision making [213]. However, the role simulation experiments can play is more than that; they are also an important part of the development of a satisfactory model. Independently whether the model is developed from scratch or it is developed based on reusing existing models, simulation experiments can play a key role in the model development process.

For example, during the construction of a model, in order to achieve high accuracy or fidelity of the model, model parameters are adjusted or tuned so that the model outputs are in satisfactory agreement to the reference system/model, i.e., parameter calibration [81, 102, 250, 263]. In this case, simulation experiments can be employed to generate model outputs for comparison.

Besides, in model validation, a crucial part of the model development process, simulation experiments are of great importance as well. To determine whether the model's output behavior has satisfactory accuracy for the intended purpose, simulation experiments can be performed with the model to generate output data for investigation and comparison. Thereby, modelers can gain important insights into the model behavior, and decide whether a valid model is obtained or further revision of this model is required. This process is also known as experimental model validation [132].

Thus, whether it is for model construction or for model validation, simulation experiments become a key tool for building models, and model development presents

itself as a process in which a model's successive refinement is interwoven with the execution of simulation experiments [208].

1.2. Motivation and Problem Statement

Despite the fact that model reuse has received a large amount of attention in modeling and simulation area and a plethora of efforts have been devoted to this endeavor, developing models via reusing existing ones is still difficult in practice and far from being well-established.

Although sharing some similarities with software reuse, model reuse may pose more difficulties for several reasons [175]. For example, Page and Opper [177] explored the computational complexity of reusing models for composition, concluding that some complexity can be introduced, such as in identifying and selecting suitable models. Due to financial reasons or the lack of contractual incentives, the motivation for the project manager to develop reusable models could be inhibited [201]. Besides, the traditional view of keeping the model as simple as possible while meeting the study objectives might also limit model reuse, e.g., due to a possible contradiction in identifying the level of abstraction for the two purposes [175, 201]. In addition, modeling is a process of abstraction and each model is built with specific objectives, assumptions, and constraints. Different users have different needs even when modeling the same system, which leads to different objectives, assumptions, and constraints. However, often this information is difficult to capture and not presented explicitly. Therefore, locating potential reusable models and recognizing incompatibilities in objectives, assumptions, and constraints become challenging. More importantly, a key issue in model reuse, and a prominent challenge as well, lies in the question of model validity, which needs to be considered as a fundamental part of any reuse strategy [190]. As mentioned earlier, models are developed with specific objectives under certain contexts, and are validated regarding those objectives and contexts. Therefore, it is well understood that due to the abstraction nature of models, no model is generally and fully valid [30, p. 5]. Consequently, when a model is reused, either for a different purpose without any modification, or for developing new models through composition or extension to meet new objectives, it is crucial to assess the model validity for the new use [190] and all valid reuse must take the objectives of reused models into account [175]. However, validation of model reuse is a non-trivial task [181]. Reusing valid models does not necessarily form a valid resulting model. The difference between the model context and objective of the reused models and that of the resulting model typically leads to changes in model behavior. Detecting those

changes and analyzing whether they are as intended, are of significant importance in order to validate the model reuse as well as the resulting model.

A great amount of work has been devoted to model reuse from different aspects, e.g., model storage, discovery and selection [56, 191, 235], reuse of generic models [74, 147], and reuse based on conceptual models [10, 19]. Some requirements in facilitating model reuse have been identified by Overstreet et al. [175]: "(1) understanding what information is needed to support reuse and how it should be represented; (2) developing mechanisms, automated and manual, to collect and record this information; (3) understanding how to design for reuse; (4) developing analysis and search tools to locate appropriate exiting components; (5) developing the ability to determine when model reuse is desirable ". Among all of these requirements, identifying the necessary information for reuse and determining its representation is an essential and also fundamental one.

In existing work for model reuse, the identified information to support reuse mainly focuses on the information about the model. In those approaches, typically the general information of the model is presented, which includes model attributes and behavior, such as model inputs/outputs or reaction rules. Examples include approaches based on DEVS (Discrete Event System Specification) [270] such as [272], approaches based on BOM (Base Object Model) [86] such as [166], approaches based on SBML (Systems Biology Markup Language) [103] such as [229], and the approach CODES (COmposable Discrete-Event scalable Simulation) [242]. In those approaches that are based on a specific formalism/standard, i.e., DEVS, BOM, or SBML, the model information is represented using the respective formalism/standard, while in the approach CODES, the language COML (COmponent Markup Language) is used. Furthermore, in order to facilitate that models can be reused meaningfully, semantic information about the model is increasingly associated. Ontologies are often used to provide a common understanding, e.g., the ontologies proposed in the BOM-based approach [167], the SBO (Systems Biology Ontology) in the SBML-based approach [45], and the COSMO ontology in CODES [242].

As discussed in Section 1.1, model development typically involves the execution of simulation experiments, and simulation experiments can be a useful tool for developing a valid model. While most of existing work for supporting model reuse concentrates on the information of the model, the information of simulation experiments is often overlooked. A model is a simplification of the real system for specific objectives and context, which have to be taken into account during the model development, especially in model validation. While the objectives and context are not directly accessible from the model definition itself, they can be reflected by simulation

experiments of the model. As pointed out by Cellier *"a model is always related to the tuple system and experiment"* [30, p. 5], model validity must be evaluated with the experiment considered, rather than the system alone; the model can be valid for one experiment and invalid for another. The importance of experiments to models has been widely recognized, and one prominent theory is the *experimental frame* proposed by Zeigler [270], which is *"a specification of the conditions under which the system is observed or experimented with"*. In recent years, annotating models with their simulation information has received some attention, such as [171]. This thesis will take a closer look at the role of simulation experiments in the model development process, and investigate how the information of simulation experiments can be exploited to support developing models based on reusing existing ones.

Although model reuse has a wide spectrum and can have different forms [165], in the modeling and simulation area, most work goes into model composition or component-based modeling, where models are reused to compose with other models in order to build larger models. However, little attention has been paid to another form of model reuse — model extension, where a previously validated model is extended to build a new one. Such extension may arise through the availability of further data and knowledge about the system under study, or through the increased understanding on the system from the part of modelers. In addition, to cope with the complexity of the model to be developed, modelers can start with building a simple, small model and extend it step by step. Similar to model composition, model reuse for extension can also help to reduce the modeling time and cost and improve the model quality, and therefore should be addressed as well.

1.3. Contributions

This work aims to support model development via model reuse, including model composition and model extension. The first key question to answer is what information is needed to support model reuse. While most of the current work focuses on information about the model, this work underlines the importance of information about simulation experiments conducted with the model.

To employ simulation experiments for developing models, firstly one needs to identify where simulation experiments can occur in the modeling process and what kind of roles they can play. Thereby, this work starts with a general introduction of simulation experiments with relevant aspects outlined, and explores the role of simulation experiments in the model development process. Based on the literature, different types of simulation experiments are discussed, such as experiments for model

calibration, experiments for sensitivity analysis, and experiments for optimization. Effective model reuse requires information of the context and objectives the reused model is built for. Simulation experiments conducted for model validation can reflect the model context as well as the desired model behavior, and therefore are of interest. When models are reused to develop new ones, the new model may possess new context and objective. However, in order for the model reuse to be meaningful, the new context and objective should be relevant to that of the reused model. Thereby, it is interesting to know what is the impact of model reuse on the model behavior and whether some key behavior exhibited by the original model is still preserved in the new model. Thus, simulation experiments conducted with the reused model can be exploited to perform experiments with the newly built model, to provide insights into the behavior of the new model and further into validity of the model reuse. Therefore, this thesis particularly focuses on simulation experiments for model validation.

Furthermore, the question of how to represent simulation experiments is covered, with a brief discussion of several existing approaches for specifying simulation experiments. An explicit, unambiguous specification of simulation experiments that is separated from model definition is a prerequisite for their reuse. Based on a comparison of different approaches, the domain specific language SESSL (Simulation Experiment Specification via a Scala Layer) [67] is selected and extended for the experiment representation. Thereby, models are annotated with the specifications of their simulation experiments, and when the models are reused, those experiment specifications can be reused as well.

Next question to answer in this work is how to use the identified information, i.e., simulation experiments, to support model reuse. Model validation via simulation experiments essentially inspects the simulation results to examine whether the model output behavior satisfies certain requirements. Inspired by *regression testing* from software development area, this work proposes to explicitly express the desired requirements on simulation results into properties as an integral part of experiment specifications. Thereby, when the experiment specification is reused for experimentation with the resulting model, experiment results can be checked against those properties to illuminate whether the model behavior satisfies desired requirements or not and whether this is as expected.

With an explicit specification of desired behavioral properties as a prerequisite, a mechanism is developed to facilitate an automated analysis of experiment results and simulation-based model checking techniques are adopted, where the expected model behavioral properties are explicitly expressed and simulation trajectories are

automatically checked using algorithms. According to the type of models and their simulation methods (e.g., deterministic or stochastic), different techniques can be integrated. For instance, statistic model checking techniques [128] can be employed for experiments with stochastic models, with probabilistic statements expressed in CSL (Continuous Stochastic Logic) [223], and properties of individual trajectories described in LTL (Linear Temporal Logic) [36] or MITL (Metric Interval Temporal Logic) [3]. To support the specification of behavioral properties and their checking, SESSL is extended in this work.

Based on SESSL experiment specifications of individual models, a mechanism is developed to automatically reuse, adapt, and execute those experiments with the newly developed model, and analyze simulation results. As a prototypical implementation of the approaches proposed in this work, TAECS (a Tool for Adaptation, Execution and Checking of Simulation experiments) is developed.

To apply the proposed approaches in facilitating model development, two types of model reuse are studied, i.e., model composition and model extension. While model composition is the focus of a substantial amount of existing work, the other type of model reuse, i.e., model extension, has received little attention. Thus, to underline the benefits of developing model through extending existing ones is also one goal of this thesis.

The main contributions of this thesis can be summarized as follows:

- exploring the role simulation experiments can play in the modeling process;
- proposing to exploit simulation experiments to support model development via model reuse, with a focus on simulation experiments for model validation;
- suggesting an unambiguous, declarative specification of simulation experiments as information representation, and proposing to explicitly describe the expected model behavioral properties as an integral part of experiment specification, together with the experiment condition to generate this behavior;
- extending the embedded domain specific language SESSL to specify simulation experiments with four aspects: model configuration, simulation configuration, data processing method, and expected model behavioral properties;
- developing a mechanism to automatically perform simulation experiments with the resulting model from model reuse, by reusing experiments of individual models;

- proposing a mechanism to automatically analyze experiment results against specified behavioral properties, with simulation-based model checking techniques integrated;
- establishing the conceptual architecture of proposed approaches and developing the tool TAECS as a prototypical implementation;
- investigating reusing simulation experiments to support two types of model reuse, i.e., model composition and model extension, and highlighting model reuse for extension;
- conducting three case studies for demonstration, one with the small Lotka-Volterra model, one based on a realistic simulation study of modeling the Wnt/β-catenin pathway, and one concerned with the receptor ligand pathway.

1.4. Terminology

In the area of modeling and simulation, there is no widely accepted, unified terminology; common terms are often used differently and the same or similar concepts are defined with different terms. In the following, some basic terms are defined for the usage of this thesis, and other relevant concepts will be explained when they first occur.

The focus of this thesis is on model development; therefore, the first key term is *model*. In [12, p. 13], a model is defined as *"a representation of a system for the purpose of studying the system"*. Furthermore, the author states that a model, on the one hand, is a simplification of the system to represent only these aspects of the system that affect the problem under investigation, and on the other hand, should be sufficiently detailed to draw valid conclusions about the system [12, p. 13]. This definition emphasizes the abstraction nature of models and that models are developed with specific objectives and assumptions. Another definition is given in [30, p. 5] (attributed to [163]) that *"a model (M) for a system (S) and an experiment (E) is anything to which E can be applied in order to answer questions about S"*. Besides, Cellier pointed out that a model is always related to the system and experiment and addressed that model validity has to take the system and especially the experiment into account [30, p. 5]. Therefore, as summarized by Leye in [131, p. 3], a model should possess three key properties, i.e., the model represents a specific system (the *reference* property), the model is built for some purpose under certain context that can be reflected by experiments (the *purpose* property), and the model only considers selected aspects of the system for that purpose (the *abstraction* property).

In addition, models can be categorized into different types, i.e., mathematical model or physical model [124, pp. 4-5], [12, p. 13], [30, p. 5]. The mathematical model is defined as *"representing a system in terms of logical and quantitative relationships that are then manipulated and changed to see how the model reacts, and thus how the system would react if the mathematical model is a valid one"* [124, p. 5]. Mathematical models can be studied in different ways, i.e., analytic solution, if applicable and computationally efficient, or simulation [124, p. 5]. This work is dedicated to the development of mathematical models that are studied by simulation, termed as *simulation models* in [124, p. 5], and henceforth referred to as models in this thesis.

The second key term in the topic of this thesis is *simulation experiment*, which comprises two basic terms *simulation* and *experiment*. In [30, pp. 4-6], definitions are given that *"a simulation is an experiment performed on a model"* (attributed to [119]) and *"an experiment is the process of extracting data from a system by exerting it through its inputs"*. In [25, p. 2], simulation is characterized as the process of *"driving the model with certain [...] inputs and observing the corresponding outputs"*. In this work, experimentation with the model is referred to as simulation experiments, which will be used interchangeably with the term experiment, unless otherwise stated. In addition to being regarded as an experiment on a given model, one common use of the term *simulation* is that, as defined in [189], *"a simulation is an execution of a model"*, which will be termed as simulation run or simulation execution in this thesis. Another use of the term simulation is related to *simulation study*, which refers to the activities that build a model of a system and conduct experiments on the model to gain information of the system or improve it. Thereby, in a simulation study, multiple simulation experiments may be performed, in each of which it may need one or more simulation runs.

1.5. Thesis Structure

The outline of this thesis is as follows. Chapter 2 presents a general introduction of simulation experiments, introduces the reproducibility and reuse of simulation experiments, and discusses the representation of simulation experiments to facilitate their reuse. Chapter 3 investigates the role simulation experiments can play in supporting model reuse with the focus laid on experiments for model validation, and presents the proposed approach with the general concept described. Chapter 4 discusses model reuse for composition and applies the proposed approach to support model composition, with two case studies presented, the one of Lotka-Volterra

models and the one of Wnt/β-catenin pathway. Chapter 5 addresses model reuse for extension and illustrates the use of the proposed approach to support developing model via extension, with a case study of receptor ligand pathway presented. Chapter 6 concludes this thesis with a summary and a brief discussion of future work.

1.6. Bibliographic Note

The discussion of existing approaches for specifying simulation experiments (Section 2.3), and the discussion of existing work for specifying model behavioral properties (Section 3.4.3), are based on the following publication, which discusses languages for simulation experiment specification.

> J. Schützel, D. Peng, A. M. Uhrmacher, and L. F. Perrone. Perspectives on Languages for Specifying Simulation Experiments. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 2836–2847, Piscataway, NJ, USA, 2014. IEEE Press.

The discussion of specifying simulation experiments with stochastic models in Section 3.4.2 is based on the following publication.

> D. Peng, T. Warnke, and A. M. Uhrmacher. Domain-specific languages for flexibly experimenting with stochastic models. *Simulation Notes Europe*, 25(2):17–22, 2015.

The requirement analysis that identifies the four aspects for simulation experiment specifications (Section 3.4), the overall architecture of the proposed approach and the prototypical tool TAECS (Section 3.5.5), have been presented in the following publication. Moreover, the idea of using the proposed approach to support model extension (Chapter 5) as well as the case study of modeling the receptor ligand pathway (Section 5.3) were also published in this paper. In the case study, the modeling work was performed by Fiete Haack, and Tom Warnke realized the implementation for $\text{MITL}_{[a,b]}$ checking algorithm, the LOESS smoothing method, and the method for replication calculation in stochastic model checking.

> D. Peng, T. Warnke, F. Haack, and A. M. Uhrmacher. Reusing simulation experiment specifications to support developing models by successive extension. *Simulation Modelling Practice and Theory*, 68:33–53, 2016.

Reusing simulation experiments to support model composition (Chapter 4) was firstly presented in the following publication, in which the first implementation of the proposed approach was realized on top of JAMES II as a proof of concept. The case study of composing Lotka-Volterra models (Section 4.4) was also described in this paper.

> D. Peng, R. Ewald, and A. M. Uhrmacher. Towards Semantic Model Composition via Experiments. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, pages 151–162, New York, NY, USA, 2014. ACM.

The challenges for composing ML-Rules models and possible methods (Chapter 4) were presented in the following publication.

> D. Peng, A. Steiniger, T. Helms, and A. M. Uhrmacher. Towards Composing ML-Rules Models. In *Proceedings of the 2013 Winter Simulation Conference*, WSC '13, pages 4010-4011, Washington, D. C., USA, 2013.

Applying the proposed approach to the case study of modeling Wnt/β-catenin pathway (Section 4.5) is presented in a paper, which has been submitted to a journal. The simulation study presented in this case study was performed by Fiete Haack, in [87, 89], where the two composed models of Wnt/β-catenin pathway had already been developed. In this paper, the model development process was replayed using the proposed approach and the developed tool TAECS.

> D. Peng, T. Warnke, F. Haack, and A. M. Uhrmacher. Reusing simulation experiment specifications in developing models by successive Composition. Submitted to the journal *SIMULATION* in July, 2016, and currently under second review.

Chapter 2

Simulation Experiments and Their Representation

With the advance in computer technology such as increased computational power and the wide use of modeling and simulation in different fields, performing simulation experiments for system analysis and decision making has become prevalent. A great many efforts have been devoted to supporting performing simulation experiments, such as experimental design and the development of efficient simulation algorithms.

In order to exploit simulation experiments for developing models, first of all, one has to identify the stages in the model development process where simulation experiments can play a role, and identify what kind of role they play. Further, the important aspects in conducting simulation experiments need to be identified so that this information can be represented and then used. Therefore, this chapter starts with an overview of simulation experiments, where the most relevant topics are discussed. As simulation experiments are increasingly receiving attention in the M&S community, a brief introduction is given on current efforts toward putting simulation experiments into a more prominent level in modeling and simulation. Finally, the representation of simulation experiments is discussed and the importance of an explicit, declarative specification of simulation experiments is underlined, which lays the foundation for the following chapter.

Section 2.1 describes the basic aspects of simulation experiments based on the literature, including the categorization of simulation experiments in Section 2.1.1, the issues that need to be considered for conducting a simulation experiment in Section 2.1.2, the structure of simulation experiments in Section 2.1.3, and the methods to support experimentation in Section 2.1.4. Furthermore, Section 2.2 discusses the idea of treating simulation experiments as first class objects, with the focus on

reproducible and reusable simulation experiments. Lastly, Section 2.3 is dedicated to the specification of simulation experiments.

2.1. An Overview on Simulation Experiments

2.1.1. Simulation Experiment Categorization

As described in Section 1.4, a simulation experiment is the process of extracting data from a model by exerting it through its inputs. Different types of simulation experiments exist, depending on how to exert the model and what kind of data are extracted. The categorization of simulation experiments can be investigated from several perspectives.

One viewpoint to categorize simulation experiments is based on the object of the experiment. Even though simulation experiments are always performed through executing the models, the object of performing experiments can be different: experimentation with the focus on studying models (hereafter termed as experiments with models) or experimentation with the focus on evaluating simulation algorithms (hereafter termed as experiments with simulation algorithms). On the one hand, simulation experiments can be performed to study the model and further to study the system (an existing one or to be designed), such as understanding the behavior of the model (system). This type of simulation experiments focuses on the model, and usually one single model is under investigation at one time to inspect the input and output relation of the model. On the other hand, simulation experiments can also be exploited for performance analysis, where simulation algorithms (implementations) are evaluated, e.g., the execution speed of the simulation algorithm, the memory consumption and the accuracy of the results [66]. Benchmark models are often used for running the experiment, and multiple models or model setups may be necessary to analyze the overall performance of a simulation algorithm [66, 80].

For the simulation experiment with models, further distinction can be made based on the stage when the simulation experiment is undertaken in the modeling and simulation process. To illustrate the roles of simulation experiments, the life cycle of modeling and simulation is employed. Figure 2.1 presents a simplified life cycle of a simulation study proposed by Balci [6]. In this life cycle, the model development phase evolves from *system and objectives definition*, through *conceptual model, communicative model*, and *programmed model*, and ends up with *experimental model*. The conceptual model is the formulation of the questions to investigate about the system; the communicative model is a representation of the conceptual model

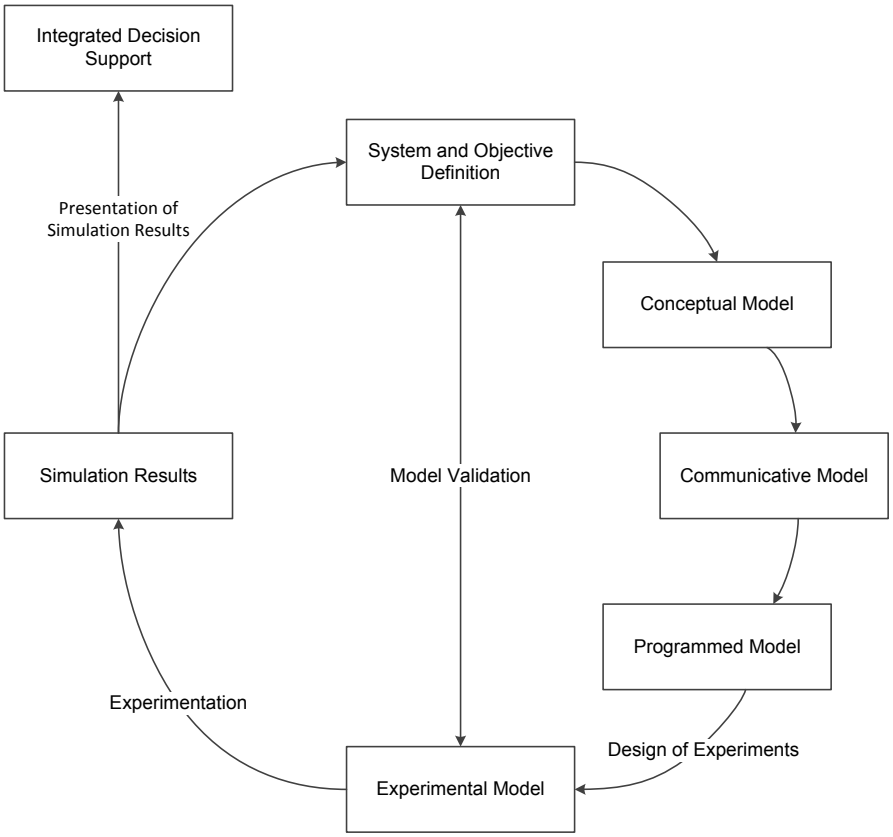


Figure 2.1.: A simplified version of the life cycle of a simulation study proposed by Balci [6]. Simulation experiments play a role both during the model development phase and after a satisfactory model is obtained.

and can be communicated with other humans, which is also identified as simulation model by the author in [6]; the programmed model is an executable simulation model and the experimental model is a programmed model incorporating an experiment plan. During the creation of models, simulation experiments can be performed for building an accurate and appropriate representation of the system. After the model development phase, i.e., a valid model has been obtained, experimentation is carried out to generate simulation results for specific purposes, such as understanding the modeled system and supporting decision making. In this phase, the built model becomes a tool, which is used to fulfill those purposes. Thereby, two types of experiments can be distinguished in the M&S life cycle, i.e., experiments for model creation such as model calibration and validation, which is during the model development phase, and experiments for answering the questions that a model is built for, which is after a satisfactory model has been developed.

In the literature, most of the work that is concerned with simulation experiments for studying models, deals with the phase when a valid model has already been attained; less emphasis is laid on simulation experiments that are conducted during

the model development phase. A discussion is presented by Kleijnen et al. [116], which illustrates that designing and performing simulation experiments during the model development process is of benefit in different ways. Performing simulation experiments can uncover some important information about the model behavior, which can provoke discussions in the modeling team over the implications of different model assumptions. Also, the gained information about the model behavior from simulation experiments can assist the modeler in forming questions, which may be difficult to capture in advance. Besides, the experiment results can rapidly reveal problems in the modeling logic and therefore provide useful insights into the direction of the model creation. What's more, conducting simulation experiments can confirm or challenge the prior expectations on the model behavior from the modeler, which is a key part of the face validation of the model.

Therefore, experiments with models can be categorized as experiments during the model development phase and experiments after a satisfactory model is built, which are also denoted as *configuration experiments* and *interpretation experiments* in [98, p. 18], respectively. While the experiments after model development are performed with a satisfactory model in order to answer the questions that the model is built for, the experiments during model development are performed with the purpose of creating a model that exhibits satisfactory behavior for the M&S objectives.

This categorization can be further looked into from another point of view, namely the goal of simulation experiments. It is of high significance to define the experiment goal, as it influences the way the experiment should be conducted [213]. According to Kleijnen et al. [116], three basic goals can be recognized in a simulation study: "*(i) developing a basic understanding of a particular simulation model or system, (ii) finding robust decisions or policies, and (iii) comparing the merits of various decisions or policies*". The goal of the simulation study is reflected on the goal of the simulation experiments performed in the study. For the first type of simulation studies, i.e., developing a basic understanding of the model or system, it covers different situations, from gaining insights into the mechanisms that are not well understood, to analyzing a validated model. In this type of simulation study, simulation experiments are conducted to explore or analyze the behavior of the model (system) after a satisfactory model is developed. To fulfill the goal of finding robust decisions or policies in a simulation study, simulation experiments can be performed for sensitivity analysis. For comparing the merits of various decisions or policies, the model is executed with different configurations in the simulation experiment and the experiment results are compared to find the optimal one.

Sacks et al. [209] identify three goals of simulation experiments: predication, optimization and calibration, and Kleijnen [115] also discusses three types of simulation experiments, i.e., for sensitivity analysis, optimization and validation of models. A classification on experiment goals is presented by Barton [16], based on the stage of modeling and simulation process as follows:

(1) early goal: validation

(2) early goal: screening variables

(3) middle goal: sensitivity analysis, understanding

(4) middle goal: predictive models

(5) middle goal: selecting the best configuration

(6) late goal: optimization, robust design

In this classification, at the early stage, simulation experiments are performed to check whether a valid model is developed, and the important variables that influence the model behavior are determined by screening. At the middle stage, simulation experiments can be conducted to quantitatively analyze and understand the model (system), such as how sensitive the model is against parameter changes, and further to provide answers to what-if questions. In the next level, some optimization of the model (system) can be achieved through performing experiments.

Based on the classification described above, Leye proposes four principal experimental goals in [131, pp. 15-16], which consist of validation, sensitivity analysis, exploratory analysis, and optimization. Validation experiments are performed to compare the model behavior to that of a reference model or system, and sensitivity analysis contains screening variables and investigates the model output with different input configurations. Whereas exploratory analysis includes conducting simulation experiments for understanding the model and providing predictions to gain insights into the modeled system, experiments for optimization execute the model with different configurations to search for the optimal one so that the model or the modeled system can be improved. Among the four types of simulation experiments, the validation experiments are undertaken during the model development phase and others occur after the development phase.

In addition, those simulation experiments that are conducted for model calibration are worth discussion. A simple definition is presented by Trucano et al. [250] that calibration is "*to adjust a set of parameters [...] so that the model agreement is maximized with respect to a set of experimental data.*" Calibration and validation

bear some similarities, i.e., both depend on comparison with data. However, the goal is different: while validation is to quantify the confidence in the predictive capability of the model, calibration is to determine the configuration of model parameters. A calibrated model needs to be further validated, for which different data should be used. Simulation experiments can be a helpful tool for model calibration, e.g., the use of experimental design approaches allows one to automatically perform experiments with systematic change of model parameter configurations. Similar to the experiment for model validation, simulation experiments for model calibration are undertaken in the phase where the model is still under development.

Based on the discussion above, a general view of categorizing simulation experiments is presented, as depicted in Figure 2.2. Simulation experiments can be categorized from three perspectives, i.e., the experimentation object (with the focus on models or simulation algorithms), the phase in the M&S process (during the model development or after the achievement of a satisfactory model), and the experiment goal (for model calibration, model validation, sensitivity analysis of the system, what-if exploratory of the system, or optimization of the system). As model development is the target of this work, only simulation experiments for studying models are discussed in detail and no further distinction is made among experiments for studying simulation algorithms.

It should be noted that the modeling and simulation process is cyclic and iterative, which means the whole process can go back and forth. Due to the abstraction and purpose property of models (as discussed in Section 1.4), the problem and objective definition can be revisited and reformulated, which may lead to a previously validated model becoming invalid, meaning a revision of the model is required. Besides, a validated model can be reused to build larger models through extension or composition, which requires going back to the model development phase again, indicated by the reference property of models. Therefore, there may not exist an absolute and exact point which separates the model development phase and the phase after that, i.e., the model development phase may not end. However, as stated by Box [24] that *"all models are wrong but some are useful"*, in a certain context and with certain questions in mind, a satisfactory model can be obtained after a careful assessment and then can be used to fulfill the purposes that the model is built for, such as understanding the modeled system and supporting decision making. Thus, the phase in which a satisfactory model is used for those purposes is referred to as after model development.

Additionally, for some simulation experiments, the goals cannot be strictly distinguished. For instance, sensitivity analysis also plays a role in the process

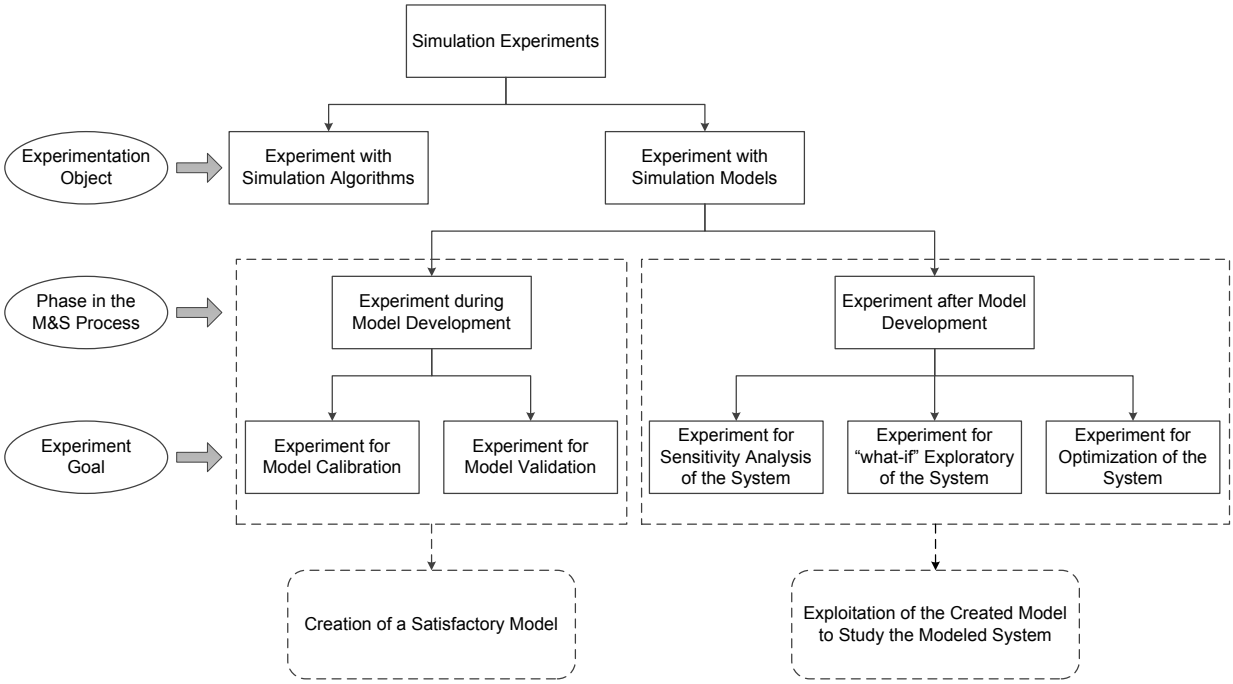


Figure 2.2.: Categorization of simulation experiments. Experiments are categorized based on three viewpoints: the experimentation object, the phase in the M&S process where simulation experiments occur, and the experiment goal.

of model calibration and validation, as it identifies the input variables that are sensitive and could cause significant changes in model behavior, and underlies the importance and priorities of those variables for calibrating and validating the model [8, 215, 250]. Moreover, the methods for optimization might also be employed for model calibration.

The categorization depicted in Figure 2.2 demonstrates that simulation experiments can play an important role not only in the phase where a validated model is obtained, to fulfill the aim of simulation study, but also during the model development phase, to assist the modeler in building valid models and accelerating the modeling process.

2.1.2. Conducting Simulation Experiments

Independent of the type of a simulation experiment, certain questions must be answered for conducting the experiment. As presented by Kelton and Barton [111], those questions include:

- *What model configuration should you run?*
- *How long should the runs be?*

- *How many runs should you make?*
- *How should you interpret and analyze the output?*
- *What's the most efficient way to make the runs?*

In addition, other questions exist, such as which simulation software (algorithm) to use, how to generate random numbers, and which results of a simulation run to record and how to record [69, 124].

Design of Experiments (DOE) for simulation is a well developed field, whose major goal is to determine the important factors using the least amount of simulating [125], with numerous work existing, [59, 110, 116, 125], to name but a few. The issue of experimental design for simulation contains two aspects, *tactical issues* and *strategic issues* [59], [113, p. 9]. According to Donohue [59], tactical issues include:

- *whether to perform a terminating or steady-state simulation,*
- *estimating the distributions of stochastic model components,*
- *selecting the initial conditions or the duration of the warm-up period,*
- *choosing the final conditions such as run time or number of events completed, and*
- *deciding on an appropriate balance between run length and the number of replications (or batches);*

strategic issues include:

- *choosing a method for the assignment of random number streams to design points, and*
- *deciding whether to use an appropriate variance reduction technique.*

In [113, p. 9], Kleijnen also lists some issues in the Design and Analysis of Simulation Experiments (DASE), tactical issues such as *the run-length of a steady-state simulation, the number of runs of a terminating simulation, and Variance Reduction Techniques (VRTs)*, and strategic issues such as *which factor combinations to simulate and how to analyze the resulting data.* There seems to be no general agreement on what the tactical issues and strategic issues comprise respectively, e.g., as presented above, the issue concerning variance reduction techniques is categorized differently by Donohue [59] (identified as strategic issue) and Kleijnen [113] (identified as tactical issue). This work does not aim to give a definitive categorization of what

are tactical issues or strategic issues. Instead, the problems that have to be considered in conducting a simulation experiment need to be identified, which can be summarized into four parts, i.e., model configuration, simulation configuration, data processing, and result analysis. To effectively and efficiently conduct simulation experiments, how to deal with those problems is of great importance.

2.1.3. Structuring a Simulation Experiment

Although simulation experiments can be performed for different goals and different methods can be used to deal with the issues described in Section 2.1.2, some general tasks exist that have to be considered when executing a simulation experiment.

Leye [131] identifies a common structure for simulation experiments, which comprises six tasks:

- *specification*, defining the experimental goals into a description so that it can be used for executing the experiments;
- *model configuration*, generating model parameter settings;
- *model execution*, executing simulation runs;
- *data collection*, collecting data during simulation runs for further processing;
- *analysis*, analyzing the results of simulation runs, which usually consists of two stages: single-run analysis and multi-run analysis.
- *result evaluation*, using analysis result to either provide feedback for creating additional interesting parameter settings, or generate final experiment results.

Accordingly, a top-bottom layered view of the structure of a simulation experiment is presented by Leye, as depicted in Figure 2.3, based on the work in [207] and [131, p. 35]. The specification of experiments, which is an essential task that has an impact on all other tasks, constitutes the topmost layer. The second layer is the multi-configuration layer, which focuses on configuring and executing the model with different parameter settings. In this layer, it starts from the model configuration task, and afterwards the model execution task, which is followed by the evaluation task. As described above, one purpose of evaluation is to create feedback for generating additional parameter settings when necessary, in which case, it goes back to configuration again, resulting in a loop; when no extra configuration is needed, final experiment results are returned. As the third layer, the multi-run layer deals with replicating simulation runs for one specific parameter setting. After the

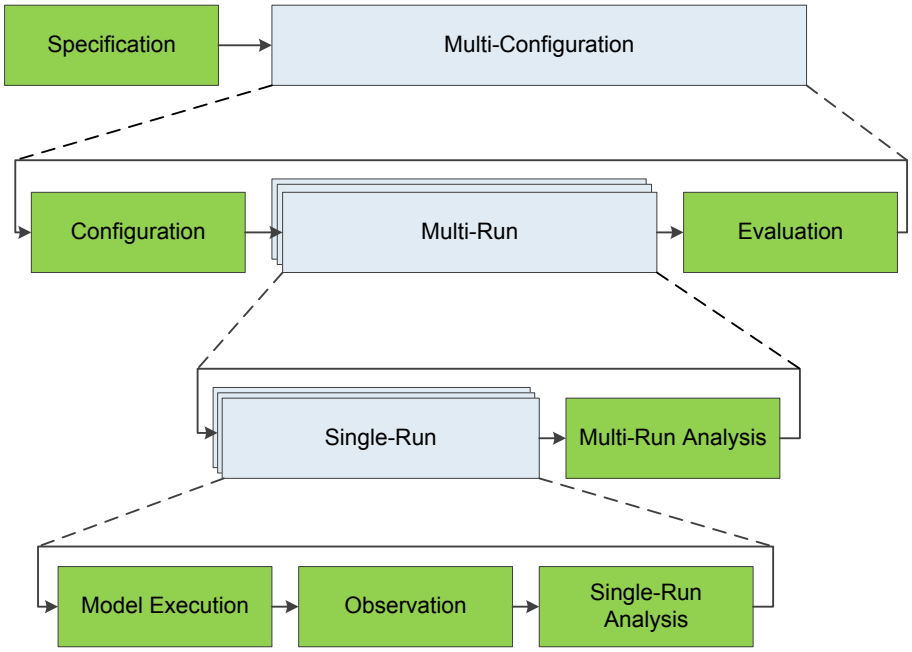


Figure 2.3.: Layered view on a common structure of a simulation experiment proposed by Leye, adapted from [207] and [131, p. 35].

scheduled replications are executed, results are analyzed to determine whether more replications are needed. The loop of replication execution and analysis is finished when sufficient data is obtained, and the results of multi-run analysis are returned to the multi-configuration layer. The single-run layer is the bottom one, where a single simulation run is executed, data is observed and collected accordingly, and the single-run analysis is performed.

2.1.4. Methods to Support Experimentation

Undoubtedly, conducting simulation experiments is a non-trivial task that may need the combination of diverse techniques. To effectively and efficiently conduct simulation experiments, knowledge about experimental methods, their use, and interpretation is needed. The benefits of providing intelligent assistance for experiments are widely recognized and have led to a variety of approaches from different aspects of the experimentation process.

Some methods provide guidance in specific application areas, often in close relation to specific simulation tools. For example, SAFE (Simulation Automation Framework for Experiments) [187], designed to be used together with ns-3, relies on typical experimental set-ups defined by experts to help novices in executing experiments with network models. AssistSim [122] aims at supporting simulation studies in manufacturing and logistics systems by exploiting domain-specific knowledge bases

to generate check lists for variables to be observed and storing experiments in a database. Also in the area of systems biology, early efforts can be found to facilitate setting up experiments, particularly for novices, e.g., COPASI (COmplex PAthway SImulator) [210], which is a platform-independent simulation tool to support different tasks of conducting simulation experiments with biochemical models.

To obviate the need to specify the entire experiment set-up in detail, some approaches support the user in selecting and configuring methods to be applied during simulation. Thereby, they often focus on specific tasks of the experimentation process, e.g., the execution of a simulation run or its analysis. MEG (MITRE Elastic Goal-Directed Simulation Framework)[179] aims to support users in executing large-scale distributed experiments on the grid, managing experiments with advanced DOE methods, and providing access to third-party tools for data processing and visualization. AutoSimOA (Automated Simulation Output Analyser) [101] is designed to provide assistance in identifying the warm-up period, estimating the number of required replications, and output analysis. A platform to execute parameter scans in the cloud is proposed by Kacsuk [104]. Helms et al. provide means to automatically, dynamically select and configure execution algorithms during a simulation run by exploiting machine learning methods [93]. In [133], Leye et al. propose a general scheme to automatically select and compose several experiment methods for a given task, such as steady state estimation.

Some tools are developed to cover the whole simulation experiment process. For instance, the tool Arena [109] is developed for discrete event modeling and simulation in applications such as manufacturing and logistics. It provides features for different experiment tasks including configuration of model parameters, model execution, data collection and analysis, and results evaluation. Various types of simulation experiments are supported, such as optimization and sensitivity analysis. MATLAB Simulink aims at the modeling and simulation of continuous and hybrid systems, and also supports various experiment tasks for different experiment goals.

All of those tools and methods provide support in the experimentation with simulation models in different manners: some of them focus on the user guidance through the experiment process, to alleviate the burden of users with well-designed graphical user interface (such as Arena and Simulink), and some aim to improve the efficiency and effectiveness of experiment execution (such as adaptive simulation [95]). However, simulation experiments performed using those methods are often not documented, at least not documented automatically within the method. The information of conducted simulation experiments is kept implicit and requires extra

efforts to extract and record, which therefore introduces difficulties in exploiting those experiments to develop models.

2.2. Simulation Experiments as First Class Objects

In modeling and simulation area, some attention has been paid to treating simulation experiments as *first class objects* [219]. The first class object is a concept from programming language design, defined as entities that can be passed as an argument, returned from a function, and assigned to a variable [220, p. 155]. In a simulation study, typically a large amount of efforts are dedicated to building simulation models. When simulation experiments are treated as first class objects, the process of conducting simulation experiments is put to a level as important as the process of developing models, and model developments and simulation experiments are jointly carried out within a simulation study. In particular, simulation experiments are managed as something solid and persisting over time, which can be reproduced, shared, and reused by third parties, rather than an implicit, fleeting process.

Reproducible Simulation Experiments

In recent years, there is an increasing emphasis on the credibility of simulation results in the modeling and simulation area. A number of work from various application fields exists, e.g., [121, 182], aiming to raise the awareness of the lack of credibility in simulation studies, i.e., the "crisis of credibility". Therefore, more and more stress is laid on the reproducibility of simulation studies, which is considered as the fundamental to obtain credible results. Pawlikowski et al. [182] state that to achieve credibility of a simulation study, the first step is to develop a valid simulation model, and the next step is to ensure that the valid model is used in valid simulation experiments. Thus, there is a rising call for publishing models together with reproducible simulation experiments, such as the guideline MIRIAM (Minimum Information Requested in the Annotation of Models) [171].

In spite of various definitions and interpretations regarding the reproducibility of simulation experiments, ranging from re-running exactly the same experiment with the same artifacts to conducting the experiment independently based only on clear statements (several additional terms are related such as repeatability and replicability), e.g., [47, 49, 126, 130, 159], an unambiguous, complete representation of simulation experiments is required to facilitate their reproducibility.

Reusable Simulation Experiments

Whereas much effort is dedicated to design and perform simulation experiments for specific purposes (as discussed in Section 2.1.1), once the designated purposes are fulfilled, those experiments are often either discarded or presented only for the publication of relevant research results. Simulation experiments are usually conducted from scratch, and are seldom reused for further study.

Whereas growing attention is paid to reproducible simulation experiments, the idea of reusing simulation experiments still stays obscure and only limited work exists. Cooper et al. [42] call for the reuse of simulation experiments (termed as "virtual experiments"), arguing that it would improve the usefulness and relevance of simulation models. The authors propose to separate experiment descriptions from model descriptions and use ontological annotations to provide semantic link between them, so that the same experiment description can be reused for multiple models and a model can be executed with different experiment descriptions. This approach is adopted in other publications of the authors [41, 43] for understanding and evaluating the functional capabilities of models, termed as "functional curation".

As identified in [42], reusable simulation experiments can be beneficial in several ways. First of all, reusable simulation experiments facilitate reproducing and sharing simulation studies. Besides, by applying different reusable experiments to a given model, it provides assistance in analysis of the model behavior under various experimental conditions. In addition, reusing the same experiment on different models supports the comparison of the behavior of those models under the same experimental condition. Further, reusable simulation experiments can also facilitate model development. For instance, when a model is developed based on simplifying a detailed one to reduce model complexity, e.g., as presented in [241], relevant experiments conducted with the detailed model can be reused to check whether the simplified model still exhibits desired behavior under corresponding experimental conditions. More importantly, model reuse can benefit from reusable simulation experiments as well. Simulation experiments are capable of carrying the information about model context and model behavior, which is of utmost importance for effective model reuse.

To facilitate a reusable simulation experiment, not only should all essential steps of the experimentation process be completely recorded in an unambiguous manner, but also the recorded experiment information needs to be easily accessible and exploited for experiment execution.

An Explicit Experiment Specification is the Prerequisite

As illustrated previously in Section 2.1.4, a great many tools and software are developed to provide assistance in the process of experimentation to make users' lives easier. However, this type of assistance usually means a certain simplification in the execution of simulation experiments, with some important parameter configuration and experimental design details hidden. These hidden details are often omitted in the published description of experiments, which therefore makes it difficult for third parties to reproduce those experiments with other simulation software and to reuse them.

Therefore, in order to manage simulation experiments as first class objects, i.e., to flexibly store, exchange, reproduce, and reuse simulation experiments, only intelligent support and guidance in the experimentation process is not sufficient; a clear and explicit specification of simulation experiments is a prerequisite.

2.3. The Specification of Simulation Experiments

2.3.1. Separating Representations of the Experiment and the Model

The idea of specifying simulation experiments separately from the description of the model, is not new and could date back to the experimental frame theory, which was firstly presented in the year of 1976 [270]. According to Zeigler et al. [273], an experimental frame is a specification of the conditions within which the system is experimented. Each experimental frame specification comprises four major parts:

- *input stimuli*: specification of the class of allowed input schedule, from which the concrete individual input samples are drawn.
- *control*: specification of the conditions under which the model will be initialized, executed, and terminated.
- *metrics*: specification of the data collection methods and the metrics for measuring the model input/output behavior, such as performance indices, goodness-of-fit criteria, and error accuracy bound.
- *analysis*: specification of the methods to analyze the collected data to draw conclusions.

The model and simulation specification language GEST presented by Ören [174] is another early work that proposes to separate experiment specification from model

definition. Defined in Backus-Naur Form (BNF), each GEST program consists of three parts, i.e., the model specification, the experiment specification and the output specification.

In a simulation study, multiple simulation experiments can be performed to the same model to accomplish different study objectives; also, the same simulation experiment may be performed in different simulation studies that share some common objectives. Therefore, separating the representation of simulation experiments from models becomes natural. With model description and experiment specification represented separately, the representation of both models and simulation experiments become more succinct and focused, which further facilitate more efficient reuse of models and reuse of simulation experiments. Thereby, standalone simulation experiment specifications are the key starting point for reusable experiments.

2.3.2. Experiment Specifications for Experimentation Support

Since performing simulation experiments is a complex task, numerous approaches exist for specifying and setting up simulation experiments to facilitate the experimentation process with models. For example, the Experimenter, the core of the Integrated Modeling Support Environment (IMSE), was a graphical tool to support experimentation with performance models [97]. Experiments are represented in *experimental plans* by the user, each of which may use one or multiple models, with one or multiple executions for each model. For each experimental plan, the Experimenter allows the specification of three types of parameters, i.e., input parameters, the control parameters and desired output parameters, and requires at least one specification for analysis to describe how to generate results from model outputs. Model execution is invoked via the Experimenter and multiple modeling paradigms are supported via embedding modeling tools. The focus of the work is on the enhancement of model exploitation.

Furthermore, a framework is presented in [228] to support performance model interoperability, which combines interchange formats for performance model, experiment specifications to allow automatic execution of experiments on the model, and the specification of outputs as well as their transformation into useful results. The Experiment Schema Extension (Ex-SE) [227] is used for experiment specification to support multiple runs of models with desired output, e.g., different model parameter configuration for different runs, the control of model execution such as iteration and alternation, and specifying desired output metrics. The Output Schema Extension

(Output-SE) and Results-SE are used to specify the format of the output metrics and the transformation from output to desired results. In [168], the general purpose language Layered Queueing eXperimenter (LQX) was proposed to define experiments for the analytic layered queueing (LQ) performance models, which supports several types of experiments such as traversal experiments, full/partial factorial experiments and sensitivity experiments.

In addition, an approach is proposed in [243, 244] to support the design and execution of simulation experiments. A tool is developed with a web-based interface to guide the user in specifying simulation experiments, which mainly comprises the specification of experiment objectives, configuration of model variables, and methods for result analysis. Intelligent support is provided to assist users in setting up experiments with DoE (Design of Experiment) principles, based on the usage of ontologies. An XML description is generated for the configured experiment and can be transformed into executable script and exported for other uses, such as reproducing and sharing the experiment.

As those approaches aim to provide support in the experimentation with models, the focus of experiment specifications is more on how to allow the specification of simulation experiment design methods so that more complex and effective simulation experiments can be performed with models.

2.3.3. Guidelines for Representing Simulation Experiments

In recent years, with the increasing call for reproducible simulation studies, many efforts have been devoted to defining standards for specifying simulation experiments.

Based on the assumption that the description of the experiment and the model are separated, the *Minimum Information About a Simulation Experiment* (MIASE) [255] describes the minimal requested information of a simulation experiment that needs to be provided in order for the experiment to be reproducible. According to MIASE, the record of a simulation experiment needs to conform to the following rules:

- *"All models used in the experiment must be identified, accessible, and fully described".* The information about the models to be included in the experiment such as the model name and location, together with the parameter configuration, must be provided.

- *"A precise description of the simulation steps and other procedures used by the experiment must be provided.".* The information about the used simulation methods, such as simulation algorithms and simulation termination conditions,

must be unambiguously described with sufficient details. Besides, the tasks performed in the experiment need to be described.

- *"All information necessary to obtain the desired numerical results must be provided."* The methods for data collection and the post-processing to generate desired final results must be defined.

Rahmandad and Sterman [193] propose a set of guidelines separately for reporting models, reporting simulation experiments, and reporting optimization results. Those guidelines are further divided into two categories, i.e., minimum requirements and preferred requirements. The *Minimum Simulation Reporting Requirements* (MSRR) refers to a detailed necessary description of simulation experiments, based on which the experiment can be repeated and reproduced to generate results that are consistent with the reported result. The MSRR includes the description of the exploited software and hardware platforms, the simulation algorithm used, the pre-processing needed on the model such as the generation of model input, the required parameter settings, the number of iterations for each scenario, and the post-processing used to produce desired results from raw output data. The *Preferred Simulation Reporting Requirements* (PSRR), beyond the minimum requirements, asks for the information that facilitates the assessment of experiment results. The information required by PSRR includes specification of sensitivity analysis, information on computational costs, information on random number generation, information on uncertainty measurement such as confidence intervals, and statistical information for experiments with stochastic models.

2.3.4. Languages for Experiment Specification

Following the efforts toward reproducibility of simulation studies and the guidelines for representing simulation experiments (as described in Section 2.3.3), various languages for simulation experiment specification have emerged during the last few years.

While *domain specific languages* (DSLs) are widely exploited for describing models, several domain specific languages have been developed to specify different aspects of a simulation experiment [219]. In contrast to general-purpose languages, which can be applied across different domains, a domain specific language is a computer language specialized to a particular problem that "speaks" the language of a specific domain [77, 251]. From the perspective of language type, domain specific languages can be divided into domain specific markup languages, domain specific modeling languages, and domain specific programming languages. In addition, two types of

domain specific languages can be distinguished: the internal (embedded) DSL and the external DSL. Whereas internal domain specific languages are implemented based on a general-purpose programming language, also called the host language, external domain specific languages are developed ground-up as independent languages and therefore require new interpreter or compilers. The key benefit of external domain specific languages is that they have their own custom syntax that is only determined by its use without constraints from the host language, while the drawback is a reduced flexibility as any new feature requires revising the syntax and corresponding parser. In contrast, internal domain specific languages inherit the constructs of their host languages and therefore require less implementation efforts from the designer. More importantly, internal domain specific languages can be easily extended by users.

The Simulation Experiment Description Markup Language (SED-ML) [256] is an XML-based markup language to encode and document simulation experiments with the information required by the guideline MIASE, to facilitate exchanging and reproducing experiments in systems biology. The SED-ML document allows the description of time-course simulation experiments and mainly contains five elements: (1) the models; (2) the simulation algorithms; (3) the tasks of the experiment; (4) the post-process of raw simulation results; (5) the output of desired results. Specifying experiments in SED-ML is independent of concrete model representation formats and simulation systems, and the Kinetic Simulation Algorithm Ontology (KiSAO) is used to provide information on simulation algorithms. However, only models that are described in XML-based languages can be supported by SED-ML. Also, simulation experiment specifications encoded in SED-ML are based on XML Schema, which makes it more machine-readable and less human-readable, and therefore requires assistance from additional tools in order to be easily used by modelers, e.g., SED-ED [1].

The ns-3 Experiment Description Language (NEDL) [90] and the SAFE Language for Experiment Description (SLED) [219] are two external domain specific languages for experiment description in network simulation. The information contained in a NEDL file allows the set-up of factorial experiments with constraints on certain parameter combination, which aims to relieve users from uninteresting design points. However, NEDL is based on XML, which leads to verbose documents that are complex to process. This motivated the replacement language SLED, which is based on JSON format.

The Simulation Experiment Specification via a Scala Layer (SESSL) is an embedded domain specific programming language for simulation experiments [67]. Based on the features of its host language Scala [172] such as meta-programming, SESSL

allows flexible experiment set-ups. By using syntactic constructs of Scala, SESSL provides the "feeling" of a simulation specification language. Serving as an additional layer between the simulation system and the user, SESSL is independent of any specific simulation system; instead, different simulation systems can be supported and integrated by creating a binding in SESSL. Through importing the binding of a simulation system, users can specify simulation experiments without interacting with the simulation system, whereas the specified simulation experiments are actually executed by this simulation system. Currently, various bindings to simulation systems exist in SESSL, for instance the binding to the modeling and simulation framework JAMES II [99]. Additional bindings can be developed without much effort.

A SESSL experiment specification can include all the information required by those guidelines for reproducible simulation experiments (as described in Section 2.3.3), such as the model location, configuration of model parameters, the simulation algorithm, the simulation stopping conditions, replication conditions, parallel execution, and data collection. Besides, SESSL provides several bindings for analysis, e.g., simulation-based optimization. In addition to its expressiveness, SESSL allows users to specify simulation experiments in a declarative style, which is more straightforward to understand and use.

A simple example of SESSL experiment is depicted in Figure 2.4. Through importing the binding denoted by `sessl.james._` (line 2), a simulation experiment is specified and executed for the modeling and simulation framework JAMES II [99]. By replacing the binding importing in the second line, the experiment specification can be switched to alternative simulation systems. As indicated by line 3, in SESSL an experiment is defined by instantiating an anonymous sub-class of the class `Experiment`. The *cake pattern* is exploited to compose the simulation experiment with different aspects, through mixing in different traits, each of which provide a specific functionality [67]. For example, to facilitate the specification for data collection and result processing such as how to record experiment output, the trait `Observation` can be added; to exploit the parallel resources available for experiment execution, the trait `ParallelExecution` is needed. Due to the features of the host language Scala, the concrete definition of an experiment is realized in the constructor of the anonymous class (line 4-12), in which each line is a function invocation, but may appear like an assignment (e.g., lines 4, 7, 11 and 12). In Line 4, the model to be used for experimentation is specified, by assigning the SESSL keyword `model` with the model location "`file-mlrj:/./SimpleModel.mlrj`", referring to a model named as "SimpleModel" and formalized in ML-Rules [153]. The configuration of model parameters is specified in Lines 5-6. Line 5 sets the value of model parameter

```
1  import sessl._
2  import sessl.james._
3  val exp = new Experiment with Observation with ParallelExecution {
4    model = "file-mlrj:/./SimpleModel.mlrj"
5    set("A" <~ 50, "B" <~ 500)
6    scan("C" <~ range(10, 1, 20))
7    simulator = SimpleSimulator()
8    stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
9    observe("A")
10   observeAt(range(0, 1, 500))
11   replications = 10
12   parallelThreads = -2
13 }
14 execute(exp)
```

Figure 2.4.: A simple example of SESSL experiment specification.

`A` and `B` to 50 and 500, respectively. In addition to configuring parameters with fixed values, SESSL allows parameter scan as well, as indicated in line 6, where the model parameter `C` is iterated over the range 10 to 20 with step size 1. The specification of the simulation algorithm to use is determined by assigning the SESSL keyword "simulator" with the simulator name, which is a defined case class in SESSL (see [67] for more detail). As depicted in line 7, the simulation algorithm implemented in the case class "SimpleSimulator" is exploited for experiments. Line 8 specifies the simulation stop conditions, i.e., the simulation stops when the wall clock time is 1 second or the simulation time is 500. To allow the specification of data collection, the functionalities provided by the trait `Observation` are needed. As shown in Lines 9-10, during the simulation, the variable `A` is observed and recorded at every time unit from time point 0 to 500. Line 11 describes the replication condition of the experiment, i.e., 10 simulation runs are carried out for each set-up. Supported by the trait `ParallelExecution`, all except 2 of available cores should be used for the experiment execution (line 12). Finally, the defined experiment is executed, as indicated by Line 14.

From the discussion above, one can see that an explicit, declarative experiment specification that is separated from model description, is able to meet the requirement of expressing all needed aspects in a simulation experiment. Not only can it support the different functionalities needed for experimentation, but can conform to the experiment representation standards outlined by existing guidelines such as MIASE. More importantly, an unambiguous specification of simulation experiments allows them to be reproduced and reused.

2.4. Summary

This chapter presented an introduction of simulation experiments based on literature, with several aspects discussed. By illustrating the categorization of simulation experiments from three perspectives, i.e., the experimentation object, the phase in the M&S process, and the experiment goal, the role that simulation experiments can play in a simulation study has been explored in Section 2.1.1. In addition to being performed with a satisfactory model to answer questions the model is built for, simulation experiments can be used to support developing the model, such as model calibration and validation. The tasks for conducting a simulation experiment have been identified, through a discussion on the issues that need to be considered in performing experiments (Section 2.1.2) and a common structure of a simulation experiment (Section 2.1.3). Following a description of existing methods to support the conduction of simulation experiments in Section 2.1.4, the idea of treating simulation experiments as first class objects was presented in Section 2.2. Specifically, reproducible and reusable simulation experiments were discussed, with a conclusion that an explicit experiment representation is prerequisite, which thereby leads to a discussion on the specification of simulation experiments in Section 2.3.

Based on a brief review of existing work on specifying simulation experiments, it has been shown that an explicit, declarative experiment specification that is separated from model description facilitates the reproducibility and reuse of simulation experiments. In particular, the domain specific language SESSL is a promising method for experiment specification.

Chapter 3

Exploiting Simulation Experiments for Developing Models via Reuse

In order to facilitate model reuse, a good understanding of the model is required, which includes not only the model definition but also the underlying context and purpose that the model is built for. In a model description that is separated from model execution, usually only the information of model structure and initial parameter configuration are presented. By solely looking at the model description, it is difficult to gain the information such as model context and assumptions. However, this information is critical for understanding and reusing the model. As discussed in Section 1.2, in current work for supporting model reuse, the focus is mainly on providing the information of model definition. In some approaches, apart from model representation in a specific formalism or language, additional information about the model semantic is provided, such as using ontologies. While it is acknowledged that simulation experiments are able to provide information about model context, making use of simulation experiments to support model reuse has not been studied in detail yet. The aim of this work is to address the exploitation of simulation experiments in developing models via reuse.

Chapter 2 has already illustrated that simulation experiments can contribute to the development of models, i.e., for model calibration and validation. Moreover, it has been demonstrated that an explicit and declarative representation of simulation experiments is a key point to facilitate reproducible and reusable experiments. Based on the discussion in the previous chapter, this chapter will present the central idea of this thesis, i.e., exploiting simulation experiments to support model development by reuse, and tackle the different aspects in realizing this idea.

3.1. Simulation Experiments in Model Development

3.1.1. Model Calibration and Validation

As discussed in Section 2.1.1, two types of simulation experiments for model development can be distinguished, i.e., those for model calibration and those for model validation (see Figure 2.2).

Model calibration is a process in which, given a set of conditions, model parameters are adjusted so that the model output closely matches the reference data (typically observed experimental data, if available) [102, 250]. When a model can produce output that is within some subjectively adequate level of precision with respect to the reference data for comparison, the model is considered successfully calibrated [118, 226]. During model calibration, the value of model parameters are optimized in order to increase the agreement between model output and reference data, and typically an objective function is selected to measure the agreement. Through minimizing the objective function, the best set(s) of parameter configuration can be obtained. Model calibration can be used to not only tune parameters, which have default or initial values, but also optimize unknown parameter values in the model, and therefore this process is also called parameter optimization or parameter estimation [226]. The first step in model calibration is to determine the selection of model parameters to calibrate, where usually sensitivity analysis is performed [4, 250]. Thereby, simulation experiments conducted for model calibration are aimed to find optimal parameter configuration, typically involving methods for optimization and sensitivity analysis.

When a model is constructed and parametrized, it has to be validated before being used to study the modeled system. A definition is presented by Trucano et al. [250], attributed to [173], that model validation "is the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model". Model validation is closely related to model verification. According to Balci [7], model validation is "substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the M&S objectives", and model verification is "substantiating that the model is transformed from one form into anther, as intended, with sufficient accuracy". Therefore, model validation is concerned with building the "right" model, while model verification is concerned with building the model "right" [7]. One interpretation of model validation is to quantify the confidence in the predictive

capability of the model for given applications and objectives through comparison with a reference model or the system under investigation [250].

A great deal of work can be found on model validation, e.g., see detailed surveys and discussions in [7, 112, 114, 215]. Different types of model validity have been introduced. For example, three levels of validity are distinguished by Zeigler et al. [271, pp. 31, 387], i.e., replicative validity, which assesses whether the model can reproduce observed behavior from the system, predictive validity, which assesses whether the model can predict unseen system behavior, and structural validity, which assesses whether the model can represent the structural relation of the system. A wide variety of validation techniques exist, and a taxonomy of more than 77 techniques classified into four main categories is presented by Balci [7], which includes informal techniques such as inspections and walkthroughs, static techniques such as cause-effect graphing and structural analysis, dynamic techniques such as comparison testing and execution monitoring, and formal techniques such as induction and proof of correctness. Among all those techniques, one category of methods depends on executing simulation experiments with the model, i.e., experimental model validation [132].

Model validation is a critical step in a simulation study, as answering the questions about the modeled system through the model, such as decision making and insight gaining, is highly influenced by whether the model is valid with respect to the system and the questions. As discussed in Section 1.2, when models are developed by reusing other models, assessing the validity of model reuse is a challenging endeavor. Also, to facilitate model reuse, it is beneficial to provide information about the context in which the model is validated. While simulation experiments are useful methods for model validation and capable of providing model context information, this work particularly targets exploiting simulation experiments that are conducted to validate models.

3.1.2. Model Validation via Simulation Experiments

As indicated by the definition of model validation (as described in previous section), the validity of models is related to the domain of applicability, which is determined by the underlying assumptions of modeling, and to the objectives of the simulation study, i.e., the questions of interest to be answered by the model. The assumptions and objectives can be reflected by experimental conditions. A set of experimental conditions incorporates the configuration of the variables that define the domain of applicability, and a model may be valid for one set of experimental conditions and invalid for another [215]. Thereby, as stated by Cellier [30, p. 5], model validation

always relates to experiments to be performed on a system as well, rather than the system alone.

Model validation via simulation experiments examines model validity by performing simulation experiments with the model. Through inspecting simulation outputs of the model, model validity at the behavioral level can be checked, which is also termed as operational validity by Sargent [215]. Depending on whether the real system is observable, i.e., whether data on the operational behavior from the system is available, different methods can be used. Based on the data availability, Sargent [215] categorizes validation techniques into comparison methods and exploration methods. Whereas in comparison methods the output behavior of the model are compared to that of the system or a reference model, exploration methods check model validity by exploring the output behavior of the model, such as parameter sensitivity analysis and graphical display. Leye et al. [132] present a list of validation techniques classified based on the tasks of conducting a simulation experiment, as shown in Table 3.1.

Experimental Validation					
Aims/Requirements	Configuration	Evaluation	Simulation	Observation	Testing
-model comparison		-metrics	-engine	-black-box	-statistical test
-degenerative tests	-factorial designs		-event queues	-gray-box	-simulation-based
-historical data	-randomized block designs		-RNG	-white -box	model-checking
-predictive data	-covariance designs			-simulation traces	
-extreme conditions	-hybrid designs				
-Turing-test	-sensitivity analysis				
-event validity	-internal validity				
-visual analytics					

Table 3.1.: Examples for different experimental validation techniques based on the specific tasks of conducting a simulation experiment, adapted from Leye et al. [132].

The output behavior of models can be examined both qualitatively, where the trends or directions of the model output behavior are examined, and quantitatively, where the magnitudes of model behavior are examined as well in addition to trends [271, pp. 370-373], [215]. Model validity can be assessed in a subjective manner, e.g., face validation, where the model behavior is checked by individuals, and turing test, which checks whether individuals can distinguish between the system and model outputs [215]. Also, statistical techniques can be employed to provide objective assessment on model validity, such as hypothesis tests and confidence intervals.

In recent years, some efforts have been devoted to bringing together model validation and formal verification techniques, which are concerned with the correctness of a system with respect to a certain formal specification or property using mathematical and logical methods, e.g., [17]. Simulation-based model checking methods have been successfully employed for model validation, in which the desired model properties are explicitly described and simulation experiments are performed to check whether the model behavior fulfills those properties, e.g., [70, 180].

Although each of those approaches checks whether the output behavior of a model meets certain requirements, the accessibility of these requirements varies, from being elusive and difficult to retrieve as they depend on interactive explorations (e.g., face validation and visual analytics [142]), through being implicitly part of the method (e.g., sensitivity analysis [60]), up to being explicitly and declaratively specified as properties in a formal language, as in the case of simulation-based model checking (e.g., in [70]).

With the focus having been laid on simulation experiments for model validation and an introduction of experimental model validation presented, subsequently the key question will be answered of how those simulation experiments can be employed to support model reuse.

3.2. Simulation Experiments and Model Reuse

It can hardly be overstated that model development is a challenging endeavor. Thereby, developing models by reusing existing ones came into play, as it carries the intuition of decreasing the required time and cost for model development. However, model reuse proves to be more difficult in practice than initially envisaged, and the validation of the resulting model as well as the model reuse pose big challenges, as discussed in Section 1.2.

Information of Model Context and Expected Behavior is Necessary

To facilitate reuse of existing developed models, providing information of those models is a prerequisite. First of all, the information of the model definition must be provided, such as in a file where the model is defined using a specific language or formalism, e.g., DEVS [270], ML-Rules [153] and SBML [103]. Secondly, in the model representation, various terms may be used to describe the same concept and the same term is often used differently, which leads to misunderstanding and increases the difficulties of reuse. For instance, both the term “NaCI” and the term “sodium chloride” can be used to refer to the salt in a model definition; however, while

interpreting the model representation, the machine will not recognize the two as the same without additional information. Thus, a common understanding and a unified terminology become a key step toward successful model reuse. Thereby, semantic information needs to be provided alongside model representation. Ontologies, which are a description of terminologies and define the concepts, relationships, and other distinctions that are relevant for the modeling domain [85, 167], are often exploited to help create a common understanding and providing an agreement on meanings of entities.

As most of current work in supporting model reuse focus on providing information of those two aspects above (also see discussion in Section 1.2, p. 5), model representation and associated semantic information only contain the information of model definition such as model structure and variables. Due to the abstract and purpose property of a model (see Section 1.4, p. 10), the context in which a model is developed and the objectives that the model is built for, have to be considered while reusing this model [175, 230]. Without knowing the context information of individual models and their expected behavior, effective model reuse is hard to achieve. However, the information of model context and expected behavior is not directly accessible from model representation and its semantic annotation. Therefore, to achieve progress in supporting model reuse, methods to provide means for representing and retrieving the context information are required.

Simulation Experiments as Context Representation

One way to capture model context is the experimental frame. The experimental frame theory was proposed by Zeigler [270] to establish the experiment conditions under which a model is valid. As described in Section 2.3, mainly four types of information are contained, i.e., model inputs, model initialization conditions, variables to observe, and data collection and analysis [50]. An experimental frame is the formulation of the objectives that motivate the development of models [271, p. 27]. The relations between model, simulator, and experimental frame are described by Zeigler et al. [271], as depicted in Figure 3.1. As one can see, the experimental frame focuses on the modeling layer, i.e., it defines what kind of experiments to execute with the model; however, the question of how the experiments should be executed also has an impact on the validation process, as illustrated by different techniques for experimental model validation (see Section 3.1.2), and therefore should be considered as well [132].

In experimental model validation, simulation experiments are performed to check the model validity. In order to conduct a simulation experiment that conforms to an experimental frame of a model, first of all, the model needs to be accessible. Besides,

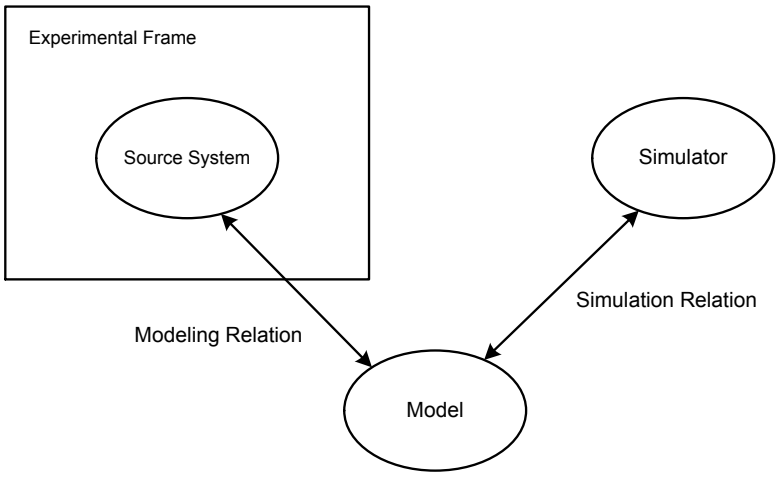


Figure 3.1.: The relations between model, simulator, and experimental frame [271, p. 26].

all experiment conditions specified by the experimental frame must be realized in the set-up of the simulation experiment, such as the configuration of model variables and variables for observation. In addition, an applicable simulation method is required for executing the experiment. What's more, the simulation output needs to be analyzed in order to check whether the model exhibits a specific behavior as desired. Thereby, a simulation experiment brings together the model, model configuration, simulation method, data to output, and result analysis. The model context and corresponding desired model behavior are reflected by the simulation experiment. Thus, simulation experiments are feasible means to represent model context and behavior information, in a more concrete and pragmatic manner than the experimental frame.

Reusing Simulation Experiments to Support Model Reuse

As illustrated in previous sections, simulation experiments performed for model validation can reflect the context information under which a model is validated and the expected model behavior. Therefore, those simulation experiments that are conducted for validating the model can be employed to support the reuse of this model.

On the one hand, by providing information of model context and behavior, simulation experiments can help the modeler determine whether an existing model is applicable to be reused for building a new model. On the other hand, when a model is reused to build a new model, the simulation experiments performed for validating the reused model can be conducted with the newly built model to check its behavior, i.e., under the same or similar experimental conditions whether the new model behaves the same as the reused model, and whether this is as expected.

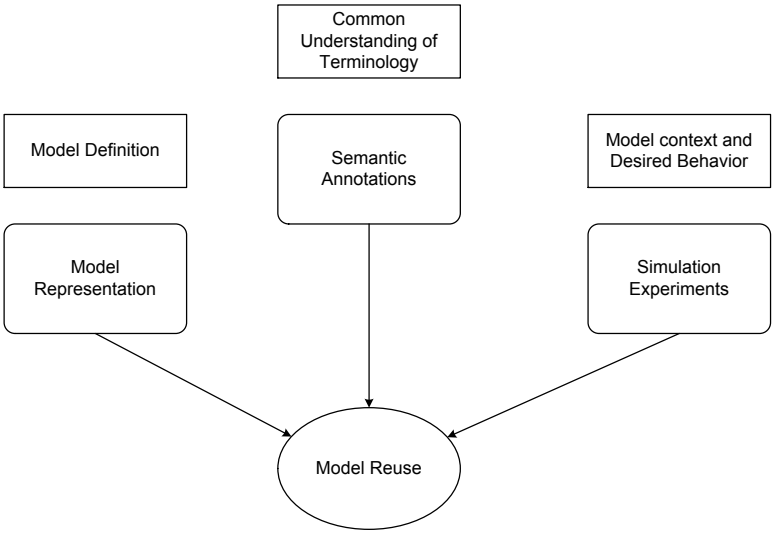


Figure 3.2.: The information needed to support model reuse.

Furthermore, when models are developed by reusing existing ones and further reused for building larger models, the history of how models are successively reused and how the model behavior evolves, is of interest and importance for the validity of the newly built models. This history can be represented by simulation experiments that are performed to validate each reused model. Thus, in complement to model representation and related semantic annotations, simulation experiments present themselves as an additional useful tool to support model reuse, as depicted in Figure 3.2.

Hence, this thesis proposes to annotate models with simulation experiments that have been executed for model validation. When a model is reused to develop new models, its simulation experiments can also be reused to perform experiments with the new models. Thereby, the behavior of the original model and the new model can be compared under the same or similar experiment conditions, to check whether the expected behavior exhibited by the original model is preserved or not in the new model and whether this is as intended. As a result, the impact of model reuse on model behavior is revealed and insights can be gained in the validity of the new model, as well as the validity of the model reuse.

3.3. Related Work

3.3.1. Reusing Simulation Experiments

Although a large body of work exists on simulation experiments, limited attention has been paid to their reuse. There exist only a few approaches that address the reuse of simulation experiments.

In the application of cellular electrophysiology modeling, an approach is presented by Cooper et al. [41], aiming to support evaluation of a mathematical model's behavior in response to multiple user-defined simulation experiment scenarios, which are termed simulation protocols by the authors. In this approach, simulation experiment specifications are designed to be independent of specific models and some commonly-used electrophysiology experiment protocols are defined. As functional curation of models is the goal of this approach, simulation experiments are reused to assess model behavior, e.g., applying the same experiment to different models or applying different experiments to the same model, and experiment results are compared. The complex post-processing of simulation outputs and modifications of model variable configuration are key points in the specification of simulation experiments.

Based on the approach proposed in [41], in [43] the authors present an on-line resource, the Cardiac Electrophysiology Web Lab, which provides a web interface that allows the characterization and comparison of electrophysiological models in a variety of experiment scenarios. The tool supports models that are in the form of ordinary differential equations and described in CellML format [136], and an extension of SED-ML presented in [41] is used for describing experiment protocols. The model description, the experiment protocol description, and the results of simulation experiments, are recorded into individual files using the COMBINE Archive format [20]. The central concept is depicted in Figure 3.3, where models and simulation experiments (protocols) are encoded separately. A simulation experiment protocol can be reused for multiple models. The focus is on analyzing and comparing behavior of different models under different experimental conditions.

Some other approaches exist to document models and simulation experiments. For example, Henkel et al. [96] propose to associate models with their semantic annotations and simulation experiments in graph databases, which aims to provide assistance in model storage and retrieval. This approach supports models that are represented in SBML and CellML formats, simulation experiments that are described in SED-ML format, and semantic annotations based on bio-ontologies. The connections between the model description, the simulation description, and the

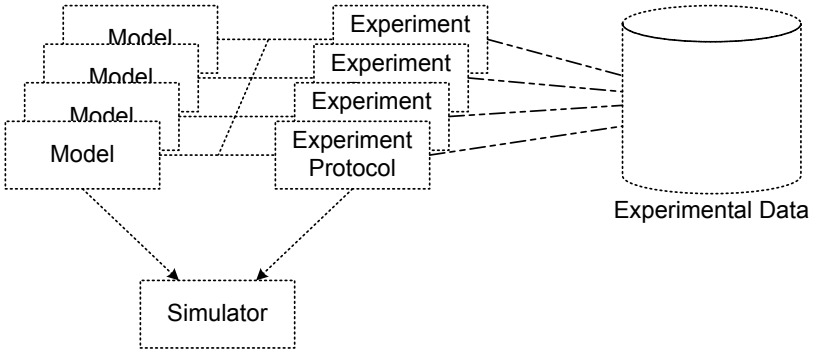


Figure 3.3.: The concept of reusing simulation experiments in the Cardiac Electrophysiology Web Lab presented by Cooper et al. [43]. Experiment protocols are described separately from model representation and may be applied to multiple models. The simulation results of applying a given experiment protocol to a model can be compared to experimental data.

semantic annotations are explicitly stored. Another approach is presented by Chreyh and Wainer [32], where the CD++ repository is designed to store models in DEVS format and their experimental frames.

In contrast to those approaches above, where simulation experiments are reused either for model curation or for model storage and sharing, in this thesis, simulation experiments are reused to support reusing existing models to develop new ones.

3.3.2. Regression Testing in Software Development

In software development, *regression testing* is used to verify that previously developed and tested software still performs correctly after some changes are made, such as software enhancements and patches [129]. When a bug is located and fixed, or a functionality is tested, it is considered good practice to record the testing procedure into a test case, which is a set of conditions under which a tester will determine whether a software is working as it was originally established to do. As the software is modified or fixed subsequently, new faults may be introduced or old faults that previously had been fixed might reoccur. Therefore, the modified software can be tested with previously created test cases to re-establish the confidence in the software [129]. Regression testing is to make sure the new change in the software has not affected any existing functionality and previously eradicated bugs remain fixed.

According to Leung and White [129], software modifications can be grouped into different types: corrective modifications, which correct or fix failures in the software so that it can work properly, and adaptive or perfective modifications, in which the

software is adapted or improved for new requirements. In adaptive or perfective modifications, usually new functionalities are introduced and the specification of the software is modified for required adaptation or improvement. Based on the modification type of software, two types of regression testing are distinguished: *progressive* regression testing, where the specification of the software is changed due to new requirements or new features added to the software, and *corrective* regression testing, where only design decisions and instructions of the software are involved and the specification remains unmodified [129]. A variety of methods have been presented to facilitate regression testing, e.g., the selection of test cases [65]. In existing work, corrective regression testing has mainly been the focus [265].

The proposed idea in this thesis bears similarities to the concept of regression testing, with model development corresponding to software development, model behavior corresponding to software functionalities, and simulation experiments corresponding to test cases. In regression testing, upon a modification of the software under development, previously created software test cases are reused to ensure that the modification does not introduce any new faults; in the approach proposed in this thesis, when a model is reused to develop new models, simulation experiments conducted with the original model are reused to inspect the behavior of the newly built model. Typically, reusing models to develop new ones may result in the change of model context and model behavior, so that some behavioral properties that used to hold for the original model would be violated in the new model. Therefore, the progressive regression testing, where both the software and its specification are modified, is more related to the proposed approach of this thesis, where a new model is developed and behavioral properties are updated.

3.4. Requirement Analysis

The essential idea of this thesis is to reuse simulation experiments of models so that when those models are reused to develop a new model, the simulation experiments of the reused models can be employed to perform experiments on the new model. To this end, an explicit, unambiguous description of simulation experiments is required, and the information that should be included in the specification of simulation experiments for model validation needs to be identified.

3.4.1. Structuring Simulation Experiments for Model Validation

In order to reproduce and reuse a simulation experiment, an unambiguous and complete experiment description is necessary. As presented in Section 2.3.3, a number of standards have been proposed for describing simulation experiments, e.g., the MIASE (Minimum Information About a Simulation Experiment) [255] and the MSRR (Minimum Simulation Reporting Requirements) [193], each comprising several aspects that should be included in the experiment specification. Moreover, in the previous chapter, the tasks that have to be considered when conducting a simulation experiment have been discussed in Section 2.1.2, and a common structure for simulation experiments was presented in Section 2.1.3. Furthermore, as illustrated in Section 3.1.2, in spite of various techniques for experimental model validation, in each of those approaches the output results of simulation experiments are analyzed to check whether the model behavior exhibits certain properties. In order to capture the information of model context and validity (see discussion in Section 3.2), the specification of simulation experiments should also incorporate the behavior properties that the model is expected to satisfy, along with the experiment conditions to generate this behavior.

Based on previous discussions and following those guidelines for experiment description, four important aspects for specifying simulation experiments are distinguished in this thesis: *model configuration*, *simulation configuration*, *data processing method* and *model behavioral property*.

- Model configuration: defining the information of the model used for the simulation experiment, including which model to experiment and how to configure the model, such as the model location, the initial state, and configuration of model parameters.

- Simulation configuration: defining the set-up of the simulation experiment, including how to execute the model, for how long and how many runs, and what data to output, e.g., simulation algorithm and stopping rules.

- Data processing method: defining the method needed to process the raw data generated by the simulation experiment, such as a smoothing method.

- Model behavioral property: defining the desired properties a model's behavior has to satisfy, which are reflected in the generated simulation trajectories. For instance, for experiments with a cell model, one property could be that the

concentration of a protein should reach a certain amount after some given simulation time.

Whereas the model configuration, simulation configuration, and model behavioral property are mandatory, the data processing method is specified only when data processing is required.

Although a common structure of simulation experiments for model validation can be derived for their specification, specifying experiments need to take into account different problems that arise from different types of simulation experiments. The problems in specifying experiments with stochastic models are of particular interest, and are discussed in the next sub-section.

3.4.2. Specifying Simulation Experiments with Stochastic Models

Depending on whether it contains random elements, a simulation model can be categorized into two types, deterministic or stochastic [124, p. 6]. Stochasticity plays an important role in many systems, e.g., in the area of systems biology. Stochastic models, such as those based on Continuous-Time Markov Chains (CTMC), are powerful means for the representation and analysis of those systems. When performing experiments with stochastic models for validation, certain issues have to be considered, as described in the following.

Probability Estimation

Due to the random factors in the model, simulation outputs of a stochastic model may differ from one simulation run to another, and therefore the simulation experiment must be replicated multiple times to gain the confidence on experiment results. As a result, the specification of multiple replications is required. Moreover, in the analysis and evaluation of the experiment results, a typical question occurs: what is the probability that the model shows a certain behavior? Thus, when specifying simulation experiments with stochastic models, the expression of probabilistic statements is necessary, as well as methods to analyze experiments results for probability estimation.

Stochastic Noise Toleration

Apart from the uncertainty of results among different simulation replications, stochasticity exists in the simulation result from a single replication as well. In each

trajectory generated in the simulation experiment, there may exist some stochastic noise, as depicted in Figure 3.4. In the simulation trajectory shown in Figure 3.4(a), the observed model variable 'x' shows an oscillation property but with stochastic fluctuations; in the simulation trajectory shown in Figure 3.4(b), the variable 'x' evolves over time, exhibiting a trend of increase, however, not strictly increasing all the time due to the stochastic noise. To effectively analyze simulation results, data preprocessing may be necessary so that the impact of the stochastic noise can be reduced.

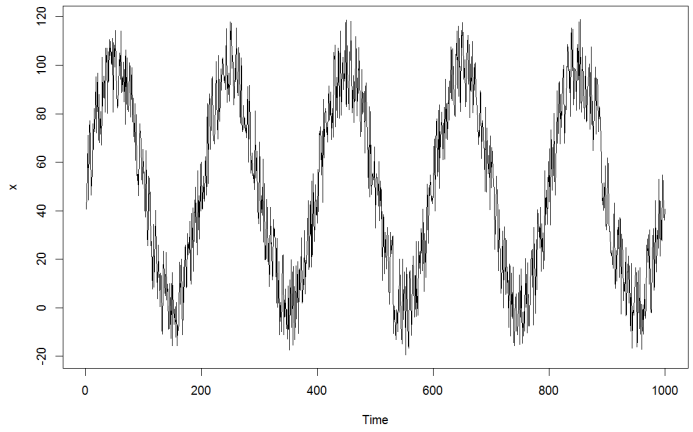
Thus, in the specification of simulation experiments with stochastic models, the stochasticity has an impact on three of the aforementioned four aspects of a simulation experiment, including simulation configuration, data processing method, and model behavioral property. First of all, multiple replications are needed in specifying experiments with stochastic models. A typical way is to fix the replication number with a constant. Alternatively, determining the replication number in a dynamic and flexible manner might be necessary. This requires that the specification of simulation configuration should support different types of replication conditions. Secondly, to deal with the stochastic noise in the simulation trajectory, the specification of data processing method should support mechanisms for noise elimination, and also properties for describing individual trajectories needs to consider the noise, e.g., to support methods for specifying noise-tolerant properties. Moreover, the specification of model behavioral property needs to allow the expression of probability estimation.

3.4.3. Specifying Expected Behavioral Properties of Models

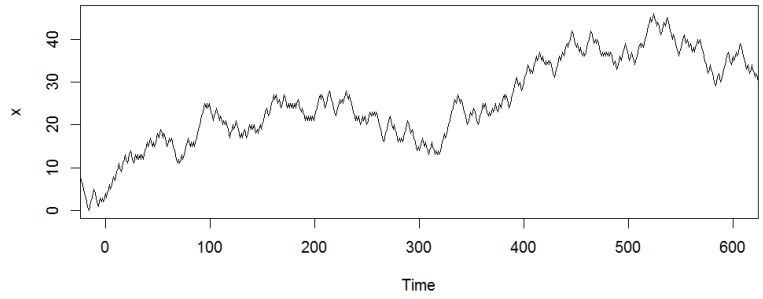
As discussed in Section 3.4.1, to reuse simulation experiments for model validation, besides the experiment conditions, i.e., the model configuration and simulation configuration, the desired model behavioral properties that should be observed through the experiments need to be specified as well.

What to Express

In experimental model validation, the evaluation of behavioral properties exhibited by a model is based on the analysis of the trajectories produced in simulation experiments. In the layered view of the simulation experiment structure (see Section 2.1.3), single-run, multi-run, and multi-configuration are distinguished. As discussed in Section 3.4.2, simulation experiments with stochastic models require multiple replications to facilitate sound output data analysis. The questions that the model is built to answer, influence whether multiple configurations of simulation experiments



(a) An example of simulation trajectory generated from experiments with a stochastic model, where the variable 'x' oscillates along the time with noise.



(b) An example of simulation trajectory generated from experiments with a stochastic model, where the variable 'x' shows a trend of increasing with noise.

Figure 3.4.: Examples of simulation trajectories generated from experiments with stochastic models. (From [185].)

are needed. Accordingly, properties that are of interest for model validation and data analysis may also involve analyzing results from a single replication, multiple replications for the same configuration, or multiple configurations.

In the case of analyzing results from one replication, one or multiple variables may be interesting for property expression, as in the example of a prey-predator model: *the prey species should either not become extinct or its population should be larger than that of its predators*. This type of properties forms the basic properties, based upon which more complex temporal properties can be expressed by relating the basic ones over time, e.g., in the property *from simulation time 0.0 to simulation time 100.0, the concentration of beta catenin should always be larger than 0*, the basic property that beta catenin concentration is larger than 0 is explicitly related to simulation time points. A temporal property may refer to not only the dynamics of one model variable, but also the temporal relation between multiple model variables, as in the example of the prey-predator model: *the number of predator starts increasing only after the number of prey grows beyond a certain threshold*. Furthermore, properties that describe a model exhibiting cyclic behavior are of interest, e.g., a model variable oscillates with a specific amplitude and period, which motivated the study of specific algorithms for detecting cycles in time series, e.g., [63]. Moreover, another common type of properties is to relate a basic property with a time period or a bounded time interval. For instance, in the property *relatively short stimulations (5 minutes or less) by tumor necrosis factor α can trigger sustained activation (60 minutes) of the NF-κB pathway in IκB-NF-κB signaling model* [192], two basic properties, i.e., stimulation by tumor necrosis factor and activation of the NF-κB pathway, are related with the time period no more than 5 minutes and 60 minutes, respectively.

When experimenting with stochastic models, as stated in Section 3.4.2, multiple replications are required for analyzing model behavior and simulation results may vary among different replications. Thus, it is crucial to quantitatively represent the probability that a certain property is satisfied by experiment results. In this case, multiple replications, rather than one specific replication, should be considered for expressing properties. The definition of the probability can refer to a property holding at discrete time points or time intervals within one trajectory, e.g., *a given property holds at least 80% of the time in a certain interval of a trajectory*. In addition, it is also of interest to define the probability of properties holding across different replications of the same configuration, e.g., *a given property holds in more than 8 replications out of 10*. More complex properties could relate properties with nested probabilities, such as *in more than 8 replications out of 10, a given property holds at least 80% of the time in a certain interval of a trajectory*.

Furthermore, expressing properties that span multiple experiment configurations would be interesting as well. For instance, the so-called "ratio-dependent theory" in the area of ecology states that if an ecosystem has richer resources, there should be higher equilibrium abundances on all trophic levels, in comparison to an ecosystem with less resources [79]. If the resources of the ecosystem are part of the initial state of the simulation experiment, then comparison of variables for more than one configuration is required. Besides, simulation experiments with the focus on sensitivity analysis and robustness also depend on comparing the results of different configurations, in which case experiments with solely a single configuration are not sufficient to provide adequate information about a model's behavior. For example, in the cell biological modeling, kinetic models with parameters can be used to describe cell dynamics. All numerical parameters need to be instantiated to a specific value before executing simulation experiments of the model. However, it may be difficult to assign a certain value to the parameters, as it may be unknown, or known without accuracy. Experiment results with single instantiation of parameters may not present adequate information about model behaviors. Thus, it is inevitable to consider uncertainties for analyzing model behavior, such as conducting robustness analysis or sensitivity analysis [26, 157]. Therefore, how to express properties for the analysis depending on multiple sets of model configuration becomes important.

Approaches to Express Properties

In the area of verification and model checking, properties that a model needs to satisfy are formally expressed and then automatically verified or checked with model checking algorithms. Generally, three approaches are used to express properties: the logic-based approach, the automata-based approach and the hybrid approach of logic and automata [23].

The logic-based approach employs logic languages to specify properties. For time related properties, a variety of temporal logics are used as specification formalisms, such as Linear Temporal Logic(LTL), Computational Tree Logic(CTL) and their extensions [36]. In addition to the standard logic operators such as "not" ($\neg$) and "or" ($\vee$), LTL provides the temporal modal operator "next" ($\boldsymbol{X}$) and "until" ($\boldsymbol{U}$), based on which the temporal operator to describe "sometimes" ($\boldsymbol{F}$) and "always" ($\boldsymbol{G}$) in the future can be derived. As a branching-time logic, CTL allows the expression of behavior with non-determinism, i.e., there exist different paths in the future, like a tree, and any one of those paths could be the actual path that is realized. Two path qualifers: "existential" ($\boldsymbol{E}$, there exists a path such that) and "universal" (($\boldsymbol{A}$), for all paths) are incorporated in CTL, in addition to the temporal operators from LTL.

To quantitatively express probabilities, the Probabilistic Computation Tree Logic (PCTL) [34, 92] and Continuous Stochastic Logic (CSL) [223] are proposed as an extension of CTL to support discrete-time and continuous-time stochastic models, respectively. In PCTL and CSL, the probabilistic path quantifier $\boldsymbol{P}$ is introduced as the replacement of the universal path quantifier $\boldsymbol{A}$ and the existential path quantifier $\boldsymbol{E}$ [248].

As an alternative to temporal logic, automata present themselves as another means for expressing properties, e.g., Büchi automata [2]. While logic-based formalisms provide high-level, concise, and expressive description languages, automata-based formalisms are more direct in modeling the dynamics of a system [23]. To combine the benefits, logics and automata are integrated together into a hybrid approach [11, 22]. Besides, regular expressions and their extensions are also used to express properties [14]. Even though automata and regular expressions have at least the same power of expressiveness as temporal logics, in general, they are perceived as too procedural and possibly low-level to be attractive formalisms for property expression [54].

Therefore, logic-based approaches are widely exploited in specifying properties for (statistical) model checking. For instance, in [152], to ensure the robustness of a synthetic genetic network, one desired property is that the fluorescence denoted as X remains below 10^3 for at least 150 min, exceeds 10^5 after at most 450 min, and needs at most 150 min for switching from low to high levels. LTL is used to express this property as:

$$\phi(t_1,t_2) = \boldsymbol{G}(t < t_1 \rightarrow X < 10^3) \wedge \boldsymbol{G}(t > t_2 \rightarrow X > 10^5) \\ \wedge (t_1 > 150) \wedge (t_2 < 450) \wedge (t_2 - t_1 < 150).$$

For stochastic models, CSL is often exploited to express properties with probabilities. One example is that in [221], a desired property is the probability of the queuing network becoming full within T time units is less than 0.5, which is formulated in CSL as:

$$\boldsymbol{P}_{<0.5}(true\ \boldsymbol{U}^{\leq T}\ full).$$

In addition, to meet different requirements of property expression, a great many variants of temporal logics have been proposed based on LTL and CTL, e.g., QFTL($\mathbb{R}$) (Quantifier-Free First-Order LTL over the Reals) [71], MITL (Metric Interval Temporal Logic) [3], and BLTL (Bounded Linear Temporal Logic) [37]. In these approaches, modal operators are employed to enrich logical formulas regarding temporal aspects, e.g., in BLTL the operator $\boldsymbol{U}^{\boldsymbol{t}}$ is defined to express that the

left argument should remain true until the right argument becomes true, which is required to happen within t time units.

While the utilization of modal operators may lower the complexity of the property expression, it also constrains the expressiveness and the possibilities to express properties with complex temporal characteristics. For example, temporal logic such as LTL and its extensions usually contains the operator "always" $\boldsymbol{G}$, e.g., $\boldsymbol{G}_{[0,t]}(d[v]/dt > 0)$ might be used to specify the property that the variable v increases from time point 0 for t time units (cf. [149] and [70]). However, as discussed in Section 3.4.2, in a trajectory produced by experiments with a stochastic model, stochastic noise exists so that during the time interval [0, t] the variable v shows a trend of increase but is not strictly increasing (e.g., as in Figure 3.4(b)), which indicates the first order derivative of the variable $d[v]/dt > 0$ cannot capture this increasing property and therefore the formula above is not applicable. This type of property definition with noise tolerance, such as the increasing trend of a variable, is common for stochastic models. Since they are not able to deal with fluctuations in the simulation trajectories that obscure the relevant properties, temporal logics are not capable of fully describing such properties of stochastic trajectories. Moreover, to deal with probability estimation, probabilistic variants such as CSL can be used to express that v increases during the interval with a probability of at least $p\%$ in terms of replications. However, to express that v increases during at least $p\%$ of the interval would be problematic.

As an alternative to modal operators, augmenting first-order logic can be used, which incorporates temporal characteristics into logical expressions with temporal arguments [225]. Two types of approaches can be distinguished: *reified temporal logics* and *Non-reified temporal logics*. While reified temporal logics rely on truth predicates that connect a non-temporal statement with time points or intervals [144] (e.g., $HOLDS[t_1,t_2,incr(v)]$), non-reified temporal logics define predicates with non-temporal and temporal arguments simultaneously [5] (e.g., $incr(t_1, t_2, v)$). In both approaches, evaluating predicates on model variables is always related to the time axis, either time points or time intervals, which facilitates an explicit and precise specification of temporal properties. Further logical connectives can be applied to model variables, time points, and time intervals. Augmenting first-order logics possess more expressive power regarding the definition of noise-tolerant descriptions, by defining predicates that allow different interpretations, e.g., $incr(t_1, t_2, v)$ can be interpreted as this predicate holds as long as the value of variable v is larger at time point t_2 than that at time point t_1, which allows the existence of stochastic noise in the trajectory. However, the complexity of first-order logic is carried over as well.

Besides, in current work model checking is typically performed against properties expressed in propositional logic; the first-order model checking problem in general has been shown to be intractable, with very little work existing on algorithms for specifications in first-order logic [84].

In recent years, with the increasing interest in annotating models along with additional information, ontologies are employed for the description of model behavior, e.g., the TEDDY (TErminology for the Description of DYnamics) [45]. TEDDY provides a more machine-readable classification of model behavior through defining vocabularies, and focuses on the most critical features of experiment results, such as increasing or oscillation. While TEDDY aims at providing semantic information of models, the property expression aims to sketchily present typical characteristics of model behavior, e.g., using the defined term *period* for characterizing a periodic oscillation and the term *limit* for the steady state behavior. Hence, in many cases, TEDDY is not able to be sufficiently precise for capturing a certain model behavior while setting it apart from irrelevant ones, especially in the case of experiments with stochastic models. Another method involving describing model behavior through representing results data is SBRML [46], which associates the model, the process applied to the model, and the data resulting from this process, based on defining ontologies in XML.

The advantages and disadvantages of these approaches are difficult to judge without knowledge of the type of properties to be described. Therefore, determining which approach to use for specifying properties is highly dependent of the specific application.

3.4.4. Specifying Data Processing Method

During the execution of simulation experiments, output data is generated and collected. However, the produced raw data may not to be directly usable for analysis, and need to be processed at first. Thereby, the specification of data processing method defines how to transform raw output data generated in the simulation experiment into the desired form for analysis, i.e., checking simulation results against the specified behavior properties.

For example, in simulation experiments with stochastic models, the raw output trajectories may contain noise; high frequency fluctuations may create difficulties for algorithms to detect patterns in the trajectory, e.g., as shown in Figure 3.4(a), or may overlay longer-term trends, e.g., as shown in Figure 3.4(b). In order to make such patterns or trends prominent also at short intervals and to allow automatic detection by algorithms, those fluctuations need to be filtered out, leading to a

smoothing process of the raw trajectories. The resulting smoothed trajectories can be evaluated against properties expressed in temporal logics without being disrupted. Other situations exist where pre-processing the raw data is necessary, e.g., the output might need to be scaled or standardized, data from different replications may need to be averaged, and also the numerical relation among different output variables could be interesting. Therefore, to meet all kinds of requirements on data processing for result analysis, the ability to integrate and specify different processing methods is required.

3.5. Proposed Approach

3.5.1. Overall Concept

The central idea of the proposed approach in this thesis is to reuse simulation experiments of individual models to support the reuse of those models for developing new models.

Model development is an intricate process, which may involve numerous simulation experiments conducted. One particular type of simulation experiments is those that are performed to gain information on the model validity. When a model is reused to develop new models, checking the validity of the new model is a crucial task, which indicates whether the original model is applicable for this reuse. Several questions arise from validating a model built through reusing other models: how the behavior of the new model is different from the reused ones; whether some key observations in the behavior of the reused models are expected to be preserved or violated in the new model, as pointed out in [231, p.4]: "Are the key behaviors of the original models still intact? Does it matter if they are not?".

According to the definition of model validation (see Section 3.1.1, pp. 36-37), model behavior and the experiment conditions, under which this behavior is exhibited, are two significant aspects for evaluating model validity. Simulation experiments not only can produce observation of the model behavior, but also carry the information of the conditions for the model to produce the behavior. Although the new model might be developed with new purposes and new context, it shares some common characteristics with the reused model; otherwise the reuse of the original model becomes irrelevant and meaningless for the development of the new one. Consequently, some behavior exhibited by the reused model should be preserved in the new model, and some may be expected to be violated. Thus, the simulation experiments that have been performed to observe the interesting behavior of the original model can be

reused to inspect the behavior of the new model, and to detect the impact of model reuse on the model behavior.

The overall concept of the approach is depicted in Figure 3.5. During the development of a model, simulation experiments are conducted for model validation. When a valid model is obtained, those simulation experiments should be described and annotated together with the model. More importantly, the specification of simulation experiments should not only include the experiment condition, but also the desired model behavioral properties, i.e., the four aspects identified in Section 3.4 (pp. 45-46): model configuration, simulation configuration, data processing method, and model behavioral property. Once this model is reused to build a new model, the specifications of simulation experiments annotated with the reused model can be reused as well. Based on those experiment specifications, simulation experiments can be performed with the newly built model. Through analyzing the simulation results against the specified behavioral properties, the impact on the model behavior of model reuse can be revealed. Hence, insights into the behavior of the new model as well as the information on the validity of the model reuse can be gained and provided to the modeler.

Therefore, an explicit specification of simulation experiments is required. In addition, a mechanism is needed to facilitate automatically reusing simulation experiment specifications of individual models, to conduct simulation experiments with the newly built model. Furthermore, a mechanism to perform automatic result analysis, i.e., checking experiment results against described behavioral properties, is needed as well. The three aspects are illustrated individually in the following sub-sections.

3.5.2. Experiment Specification in SESSL

According to the discussion presented in Section 2.3, an explicit, declarative descriptions of simulation experiments is beneficial for reproducing and reusing experiments. As illustrated in Section 3.4.1, the proposed approach in this thesis requires an explicit specification of behavioral properties, in addition to corresponding experiment conditions. Despite the fact that a number of approaches have been proposed for specifying simulation experiments, e.g., SED-ML (see Section 2.3.4, p. 30), none of them was designed to explicitly describe expected model behavioral properties that are observed from the specified simulation experiment.

SESSL is an internal domain specific language to specify simulation experiments in a declarative style, as described in Section 2.3.4 (pp. 31-32). As SESSL was initially designed to support users in setting up and executing simulation experiments,

experiment specifications in SESSL are unambiguous and executable. Apart from all the features SESSL provides, e.g., the ability to specify model configuration and simulation configuration and to support analysis such as optimization, a more important aspect for this thesis is that, as an embedded domain specific language, SESSL can be easily extended with new features. Therefore, SESSL is selected for experiment specification in this work.

As identified in Section 3.4.1, the specification of a simulation experiment should incorporate four aspects, i.e., model configuration, simulation configuration, data processing method and model behavioral property. SESSL originally supports specifying the first two aspects, i.e., model configuration and simulation configuration, and thus additional extensions are needed to allow specifying data processing methods, model behavioral properties, and those methods required to facilitate automatic property checking. Thereby, SESSL was extended with a new trait `Hypothesis` implemented. This trait can be integrated into the class `Experiment` as a mix-in to provide corresponding features (see discussion in Section 2.3.4, pp. 31-32).

For illustration, a simple example is depicted in Figure 3.6, based on the experiment presented in Figure 2.4, which specifies a simulation experiment for the modeling and simulation framework JAMES II by importing the corresponding binding (line 2). In line 3, an experiment is defined that supports observation, parallel execution, as well as property specification and checking. Lines 5-7 depict the model configuration, i.e., the model used for experimentation (a ML-Rules model named `SimpleModel`) and the assignment of three model parameters (`A`, `B`, and `C`). The simulation configuration is shown in lines 9-14, which includes the simulator to use (line 9), the simulation stop condition (line 10), the variable to observe during simulation (line 11) and how to observe (line 12), the number of replication for each configuration (line 13), and the computer resource to use for parallel execution (line 14). Line 16 specifies the data processing method, where the method named `SimpleDataProcessor` is used. Lines 18-19 are the specification of model behavioral property, starting with the SESSL keyword `assume` (line 18). As depicted in line 19, a property based on LTL is defined to state that the variable `A` should be always larger than zero, using the operator "Always" $\boldsymbol{G}$.

In this example, for the purpose of illustration, the constructed data processing method `SimpleDataProcessor` is used in the experiment specification, and a straightforward property expressed in LTL is specified. However, the proposed approach is open to various methods for data processing and behavioral property specification, which can be integrated into SESSL without much effort. The selection

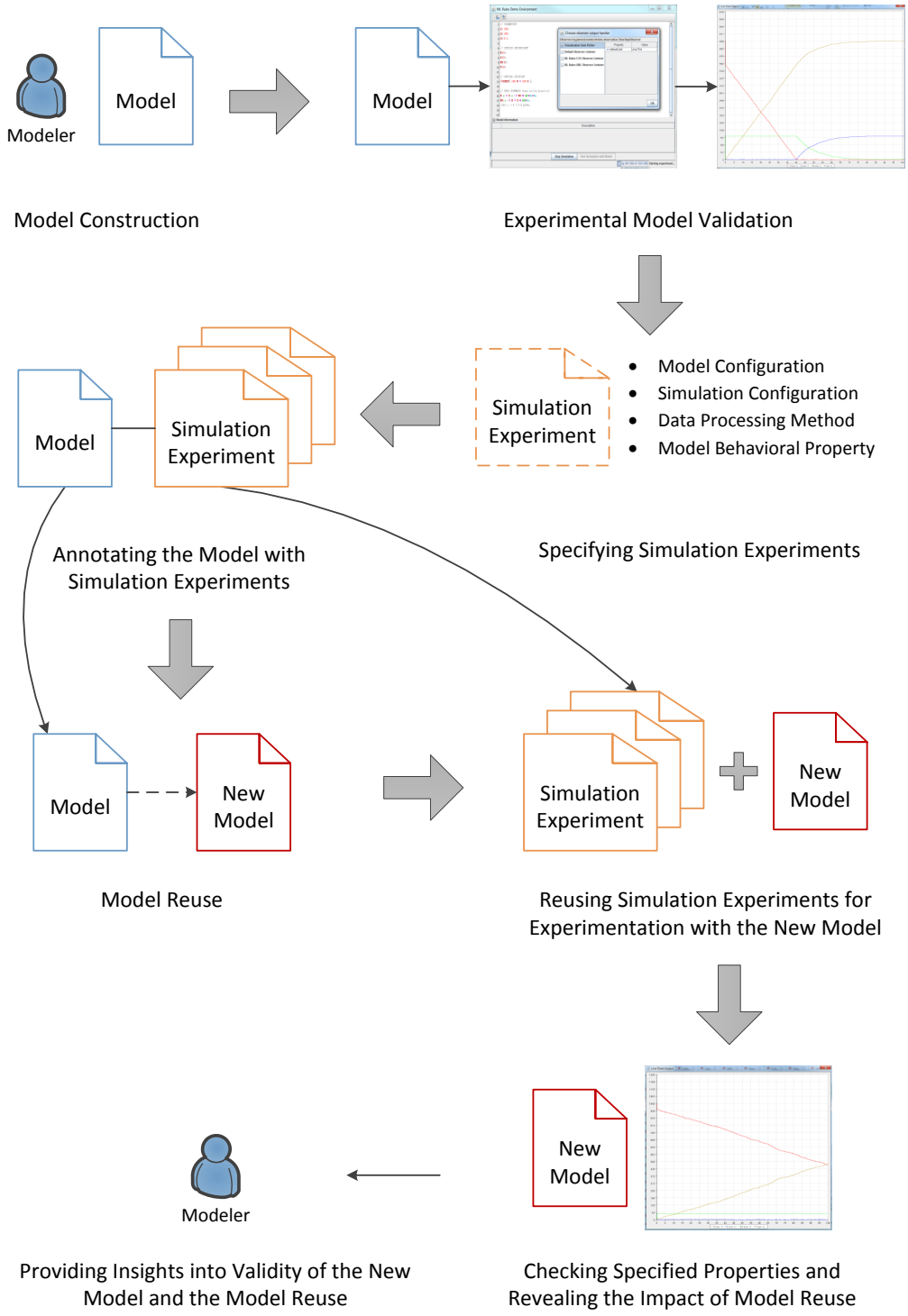


Figure 3.5.: The overall concept of the proposed Approach. Models are annotated with their simulation experiments. When models are reused to develop new models, those simulation experiments are reused for experimentation with the newly built model. Simulation results are analyzed to check whether the specified behavioral properties that used to hold for the reused model, are satisfied by the new model, and whether this is as intended. Thus, some insights can be provided into the behavior of the new model as well as the validity of the model reuse.

```
import sessl._
import sessl.james._
val exp = new Experiment extends Observation with ParallelExecution with Hypothesis {
  //model configuration
  model = "file-mlrj:/./SimpleModel.mlrj"
  set("A" <~ 50, "B" <~ 500)
  scan("C" <~ range(10, 1, 20))
  //simulation configuration
  simulator = SimpleSimulator()
  stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
  observe("A")
  observeAt(range(0, 1, 500))
  replications = 10
  parallelThreads = -2
  //data processing method
  dataProcessor = SimpleDataProcessor()
  //model behavioral property
  assume(
   G(variable("A") > 0))
  }
```

Figure 3.6.: An illustrative example of SESSL experiment specification with four aspects: model configuration, simulation configuration, data processing method and model behavioral property.

of methods to use depends on concrete applications, which will be illustrated in later chapters within case studies in more detail.

It is notable that in a SESSL experiment specification, once specified by the user, the configuration of model parameters is completely determined, and the simulation experiment is executed with this configuration. In addition, in the model file a full configuration of model parameters should be provided, with default values assigned to each model parameter. When some assignments of model parameters are explicitly configured in the experiment specification, which could be different from the default values in the model file, those assignments are used for the experiment execution; otherwise, the experiment is executed with the default values provided in the model file.

3.5.3. Experiment Adaptation, Generation and Execution

When the experiment specifications of individual models are reused, those specifications may not be directly usable for performing experiments with the newly built model and certain adaptations are necessary.

First of all, seeing that the experiment generation for the new model is based on the experiment specifications of individual models, to reuse those experiment

specifications, configuration of the model needs to be adapted. In other work that generates simulation experiments from description scripts (e.g., SED-ML and NEDL), typically only one model is involved, i.e., the model that experiments are generated and executed with is the same as the one documented in the experiment description [90, 256]. However, when experiment specifications of individual models are reused and applied to the newly built model, experiments are generated for a different model (i.e., the new model), rather than the model described in the experiment specification (i.e., the reused model). Thus, the model location needs to be updated from the original model to the new one. Besides, configuration of model parameters may also need to be adapted. For instance, when new parameters are added in the newly built model, configuration of their assignments is required for executing the experiment; however, those parameters are not part of the original model and no configuration is included in the experiment specification of the original model as well. Therefore, additional information is needed for the configuration of new parameters, either by providing assignments in the new model file or by specifying it through experiment specifications, which requires an update in the reused experiment specifications of the original model. In addition, the configuration of some existing parameters might need to be changed, possible reasons including updated knowledge from the modeler's side, requirements from the context or objectives of the new model, incompatibility resulting from the model reuse, and so forth. Furthermore, adaptation in the model behavioral properties of the experiment specification may be necessary as well, e.g., some model variables are renamed during the construction of the new model, in which case the variables specified in the observation conditions also need to be renamed accordingly. All the information that are required for adapting experiment specifications should be defined in the *adaptation information*.

An example of adapting experiment specification is presented in the following, based on the previously described example shown in Figure 3.6. The model named as `SimpleModel` is annotated with its experiment, and this model is reused by the modeler to build a more complex model, named as `NewSimpleModel`, which contains a new model variable `D`. Therefore, the newly built model has three variable from the original model, i.e., `A`, `B`, `C`, and the newly introduced variable `D`. During the construction of the new model, in order to resolve the name conflict between variable `A` and another model variable, which may be from a second reused model and shares the same variable name, the modeler renames the variable `A` from the `SimpleModel` into `newA`. In addition, the assignment of some parameters is adjusted, e.g., decreasing `B` from 500 to 200. Meanwhile, the initial value of the new model variable needs to

```
1  import sessl._
2  import sessl.james._
3  val exp = new Experiment extends Observation with ParallelExecution with Hypothesis {
4    //model configuration
5    model = "file-mlrj:/./NewSimpleModel.mlrj" /** "file-mlrj:/./SimpleModel.mlrj" */
6    set("newA" <~ 50, /** "A" <~ 50 */
7        "B" <~ 200, /** "B" <~ 500 */
8        "D" <~ 30) /** not included in the original specification*/
9    scan("C" <~ range(10, 1, 20))
10   //simulation configuration
11   simulator = SimpleSimulator()
12   stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
13   observe("newA") /** observe("A") */
14   observeAt(range(0, 1, 500))
15   replications = 10
16   parallelThreads = -2
17   //data processing method
18   dataProcessor = SimpleDataProcessor()
19   //model behavioral property
20   assume(
21     G(variable("newA") > 0) /** G(variable("A") > 0) */
22   )}
```

Figure 3.7.: The adapted SESSL experiment based on the experiment described in Figure 3.6. The adapted information is followed by the original specification in the same line (comments in gray).

be configured, e.g., setting `D` to 30. All those changes are recorded in the adaptation information.

Thereby, in order to perform experiments with the new model by reusing the experiment of the original model, i.e., the one presented in Figure 3.6, an adaptation of this experiment specification is required. Based on the defined adaptation information, the adapted experiment specification, which can be executed with the new model, is as shown in Figure 3.7. The model location is updated from the original model to the new model `file-mlrj:/./NewSimpleModel.mlrj` (line 5). The model parameter `A` needs to be renamed as `newA`, which leads to an update in model configuration (line 6), observation condition (line 13) and property (line 21). The assignment of parameter `B` is updated to 200 (line 7) and the configuration of new parameter `D` is added (line 8). In the experiment specification, the rest of model configuration as well as simulation configuration stays unchanged, and the data processing method is reused as it is.

Various ways exist to obtain the information needed for adapting the experiment. For example, adaptation information can be automatically generated by monitoring and analyzing changes in model files, in which case the work on model transformations

(e.g., [161]) and on model version control (e.g., [257]) are of relevance. Currently, in the prototypical implementation of this work, all the information needed for adapting the experiment is specified by the user. Since experiments specified in SESSL are executable, once experiment specifications of the original model are adapted, they can be directly applied to the new model and experiments can be executed.

Assume that a new model m' is created by reusing existing models. E is a set of experiments that have been executed with the original models. Each experiment $e \in E$ contains a model configuration including model location e_m and parameter assignment e_a, a simulation configuration including observation conditions e_o, a data processing method e_d, a model behavioral property e_p and the replication condition e_{rep}. Taking the experiment presented in Figure 3.6 as example, e_m denotes the model `file-mlrj:/./SimpleModel.mlrj` (line 5) and e_a refers to the configuration of three parameters `A`, `B` and `C` (line 6-8). Lines 12-13 constitute observation condition e_o, which includes the variable to be observed (i.e., `A`) and how to observe (i.e., at each time point from time 0 to 500). The method denoted as `SimpleDataProcessor` is the data processing method e_d (line 17) and behavioral property e_p is depicted in lines 19-21. The replication condition e_{rep} is configured as 10 (line 14).

In the adaptation information κ, users can specify the change of parameter configuration κ_a and the change of model variables κ_{names}. In the example of adapting experiment described in Figure 3.7, κ_a includes assigning parameter `B` to 200 and `D` to 30, and κ_{names} is a mapping from variable name `A` to `newA`. Additionally, users can specify the expected result of checking the property on the new model m' in the result expectation r, which defines that for each experiment e in E the property e_p will hold or not.

The procedure of adapting, generating and executing experiments is depicted in Algorithm 1. The algorithm iterates all experiments of reused models. For each experiment $e \in E$, first of all, the experiment is adapted based on the adaptation information κ (lines 9-16). The model location e_m is updated to the new model (line 10). The configuration of model parameters e_a is updated as well, including the assignment of parameter values and parameter names (line 12). Besides, the variables in the property e_p and in observation conditions e_o are renamed accordingly (line 14). Apart from necessary adaptation, all the other aspects of the experiment, e.g., the simulator, simulation stop time and the data processing method e_d, remain the same. Subsequently, a new experiment e' is generated (line 16).

In stochastic simulation, replication numbers have an important impact on simulation results, and therefore influence the evaluation of properties. To gain confidence in checking the probabilistic properties of the extended model, a certain strategy for

Algorithm 1 Algorithm for reusing experiment specifications.

m': the new model built by reusing other models.

E: all experiments executed for reused models.

κ: adaptation information.

r: result expectation.

```
1  // Test result
2  res ← true
3  // Return set for successful experiments
4  E_+ ← ∅
5  // Return set for failed experiments
6  E_− ← ∅
7  for each experiment e ∈ E
8  /*** Experiment adaptation */
9  // Update model location to the new model
10   e_m' ← updateModel(e_m, m')
11 // Update model configuration, including parameter assignment and renaming
12   e_a' ← updateParameterConfig(e, E, κ_a, κ_names)
13 // Rename variables in property and observation variables
14   e_o', e_p' ← renameVariables(e, κ_names)
15 // Generate experiment for the new model
16   e' ← generateExperiment(e_m', e_a', e_o', e_p', e)
17 /*** Experiment execution */
18 // Determine replication number
19   e'_rep ← calculateReplicationNumber()
20 // Execute experiment
21   Y ← run(e')
22 /*** Data Processing */
23   Y' ← processData(Y, e'_d)
24 /*** Property checking */
25   result ← check(e'_p, Y')
26 /*** Result return */
27 // Add successful experiment
28   if(result is true)
29     E_+ ← E_+ ∪ {e'}
30 // Add failed experiment
31   else
32    E_− ← E_− ∪ {e'}
33 // Compare and aggregate adherence to result expectation
34   res ← res ∧ (result equal getExpectation(r, e))
35 end for
36 return res, E_+, E_−
```

determining the required replication number of experiments is necessary, which is implemented in the function `calculateReplicationNumber` (line 19). Afterwards, the new experiment is executed, resulting in output trajectories Y (line 21). New trajectories Y' are obtained by applying the data processing method e'_d on the trajectories Y (line 23). The new trajectories Y' are checked against the property e'_p(line 25) and the checking result *result* is either true or false. Independent of whether the property is checked successfully or not, the experiment specification is recorded accordingly (lines 27-32). Further, the checking result is compared with the result expectation *result*$'$ specified by the user, and based on aggregating the comparison outcome, the test result *res* is formed with a value of true or false (line 34).

When all experiment specifications are adapted, executed and checked, two types of information are returned to the user. On the one hand, the result expectation specified by the user is tested against the checking result for each experiment specification, and the test result is returned to the user, which corresponds to the *res*. When the checking results of all experiments are as expected, the test result is true; if the checking result of one experiment does not match the expectation, false will be returned (as depicted in line 34). Since the result expectation is derived by the user based on how the new model is created, with this type of information, the user can find out whether the new model actually behaves as expected, and if not, the reuse of the original models may need to be revised. On the other hand, regardless of result expectations, each experiment specification along with its checking result are recorded and returned, corresponding to $E+$ and $E-$ (lines 27-32). Those experiment specifications that have been checked successfully are returned, which are annotated with this model for further reuse; those whose checking result is false are returned to the user, either to be revised or discarded.

The procedure of adapting experiment specifications may vary depending on the type of model reuse, especially regarding parameter configuration. For instance, in model composition, when two models are composed together, typically the composed model has a higher parameter dimension than the reused models, i.e., it contains all the model parameters from both models. The two reused models may share common model parameters but with different configurations, and therefore a certain strategy is needed to resolve the conflict in the configuration of those common model parameters. But this conflict resolution is not required in the case of model extension, as only one model is reused. The adaptation of experiment specifications will be studied in more detail in combination with specific type of model reuse in the following

chapters, where reusing models for model composition and extension are investigated, respectively.

3.5.4. Result Analysis

In the field of verification and model checking, desired properties are formally expressed using languages such as temporal logic, and further algorithms are used to automatically check whether those properties are satisfied. Model checking is a well-established technique with a large body of work existing and a variety of tools developed, e.g., BIOCHAM (the BIOCHemical Abstract Machine) [28], PRISM (a probabilistic model checker) [100], MARCIE (the Markov Reward Model Checker) [108]. More information can be found in textbooks and detailed surveys such as [27, 29, 36, 128].

As model checking techniques evolved in recent decades, simulation-based approaches have been employed for checking specified properties, in addition to numerical and symbolic analysis approaches. A typical use of simulation-based model checking techniques is to solve the probability model checking problem, which is "given a property ϕ, checking whether a stochastic system satisfies ϕ with a probability greater than or equal to a certain threshold θ" [35]. Numerical approaches for solving this problem compute the exact probability for all the executions of the system, which are memory intensive and may not scale up for systems with large state spaces [37]. In contrast, simulation-based approaches for probability model checking problem, also known as statistical model checking (SMC), rely on executing simulation to generate finite samples from the system, evaluating each sampled trajectory against the given property, and answering the question whether the samples provide statistical evidence to show the system satisfies the given property with a certain probability [35, 107, 128]. Thereby, in those approaches, apart from mechanisms for estimating probability, methods for checking individual simulation trajectories against a given property specification are also needed [35, 266]. Hence, the work concerned with property monitoring or runtime verification is relevant, where individual simulation trajectories are analyzed [18, 150, 170]. Moreover, in the literature, many algorithms have been developed for analyzing produced simulation trajectories against properties specified with formal languages such as LTL, e.g., [71].

Following the concept of model checking techniques, this work aims to support an automatic result analysis as well, i.e., by making use of algorithms to automatically check experiment results against the specified model behavioral properties. As discussed in Section 3.4.3, the presented approach in this thesis is independent of specific methods for specifying model behavioral properties. With simulation

experiments specified in SESSL, which allows easy extension, different methods for property checking can be supported and integrated. When languages such as widely used temporal logics are used for property specification, the existing checking algorithms can be exploited and integrated in the proposed approach, as well as newly developed algorithms. In addition, other than existing techniques, new methods such as a new language for specifying properties can also be supported by integrating into SESSL, e.g., [260].

Furthermore, as stated earlier, based on how the new model is created from reusing others, the modeler can specify the result expectation of checking behavioral properties on the new model, i.e., which properties are expected to still hold and which should be violated. Thereby, after experiment specifications are adapted and executed for the newly built model, experiment results are checked by property checking algorithms, and the checking results are automatically compared to given result expectations.

3.5.5. General Architecture

The overall conceptual structure of the presented approach is depicted in Figure 3.8, which is divided into three layers based on the three main elements of the approach, i.e., reusable experiment specifications, experiment generation from specifications, and result analysis.

To reuse simulation experiments of individual models for conducting experiments with the model built by reusing those models, at first the model to experiment on (i.e., the newly built model) and the simulation experiments to reuse must be accessible. Thereby, in the topmost layer, a user interface should be provided, where users can specify the model to experiment on and the simulation experiment specifications to use. Also, users should be allowed to define necessary information for experiment adaptation when experiment specifications need to be adapted, and the result expectation for property checking. All of this information is handed over to the second layer. Different types of user interface are possible, such as a graphical user interface, where users are able to load the model and experiment specifications, and also to specify adaptation information and result expectation via the interface.

The second layer deals with the experimentation. Based on the information passed over by the first layer, the second layer performs experiments with the model based on given experiment specifications. Those experiment specifications are adapted first when necessary. After the experiment execution, corresponding data process methods in the experiment specification are applied to the raw simulation output. When the simulation output is applicable for analysis, the second layer triggers the

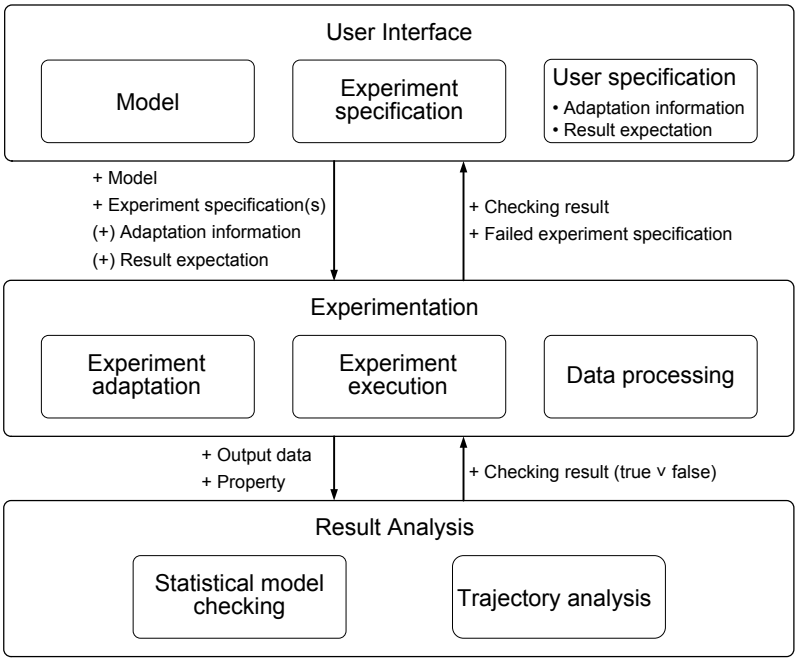


Figure 3.8.: The overall architecture of the proposed approach. As the user interface, a SESSL extension is used in the upper layer where the user can specify the model, experiment specifications, adaptation information and result expectation. The middle layer conducts experiments and performs data processing, with two types of input, either the model, experiment specifications, adaptation information and result expectation, or only the model with the experiment specification. With output data and property passed by the second layer, the bottom layer checks the property and returns checking results. [Adapted from [186]]

third layer to perform property checking, with the formalized properties and output data handed over.

The bottom layer is concerned with analyzing results. In this layer, the output data is checked against the desired properties. The checking results are returned to the second layer, where further analysis may be performed, e.g., in the case of statistical model checking, to see whether more replications are required. When no additional experiments are needed and the evaluation of experiment outputs is finished, the results are compared with result expectation if available. Finally, the overall results are returned to the user.

The prototypical implementation of the proposed approach has been designed and realized. For the first layer, the extended SESSL with the `Hypothesis` trait (see Section 3.5.2) is used as the user interface. Besides, a graphical user interface is under development (see [205]).

As a proof of concept, the first implementation was realized on top of the modeling and simulation framework JAMES II, where SESSL is the user interface, experiment generation and execution rely on the experimentation layer of JAMES II, and checking algorithms for result analysis are integrated into JAMES II. Although JAMES II provides the approach with the applicability to various modeling formalisms and the flexibility for further extensions, experiment adaptation and execution are realized by transforming SESSL specifications into JAMES II experiments, and the adapted experiments are internally generated within JAMES II and not accessible. Consequently, those adapted experiments, which are actually executed with the new model, need to be manually specified by the user into SESSL specifications, and then to be annotated with the new model if checked successfully.

To alleviate this problem and provide further user assistance, the TAECS (a Tool for Adaptation, Execution and Checking of Simulation Experiments) was developed based on SESSL, which consists of the middle layer and the lower layer of the general architecture in Figure 3.8, while SESSL is the user interface as well. As SESSL allows specifying simulation experiments for different simulation systems through creating bindings, exchanging modeling formalisms and simulation methods are implicitly supported by the approach. Similarly, the approach is open to different languages for property specification and their checking algorithms, which should be integrated into SESSL.

In addition to performing automatic experimentation by reusing experiment specifications, the tool TAECS can also allow the user to directly executing experiments on models, or revise the experiments that fail in the checking. In both situations, an experiment specification and the related model are passed to TAECS, without adaptation on the experiment specification. Thereby, two working processes are distinguished in TAECS, as shown in Figure 3.9. In case A, experiment specifications are adapted firstly, based on the specified adaptation information. This case corresponds to reusing experiment specifications for developing new model, i.e., the core concept of this work, and the procedure is as presented in Algorithm 1. In case B, TAECS provides assistance in conducting experiments with models and documentation of experiments, where no experiment adaptation is needed. Taking only the experiment specification and the model as input, the algorithm for the procedure in case B is similar to Algorithm 1, without the experiment adaptation and comparison to result expectation.

For each of the three case studies in this work, which will be presented in Section 4.4, Section 4.5, and Section 5.3, both working processes were used. Before being annotated to individual models, all experiment specifications were tested with the

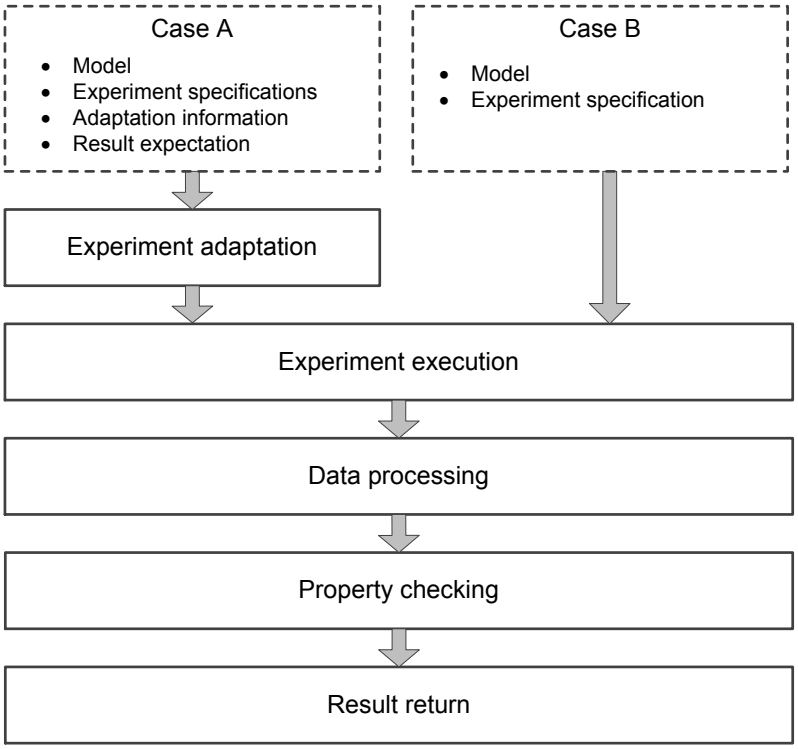


Figure 3.9.: The working process of TAECS with two cases. In both case A and B, TAECS executes experiments based on experiment specifications. And then corresponding data processing is performed on the raw experiment results. Afterwards, the processed data are checked against specified properties. In case A, TAECS takes experiment specifications, the model, adaptation information and result expectation as input, and the experiment specifications are first adapted, while in case B it only takes the experiment specifications and the model as input, and the experiment specifications are directly executed and checked without adaptation. (From [186])

model using the working process in Case B. When experiment specifications of individual models were reused, the working process in Case A was used to test experiment specifications on the new model.

3.6. Summary

This chapter presented the general concept of employing simulation experiments to support model development by reuse. At first, through a discussion on the different types of simulation experiments that are performed during model development, Section 3.1 narrowed the focus of this thesis to the experiments for model validation, and further described different techniques for validation via simulation experiments.

Subsequently, Section 3.2 introduced the idea of reusing simulation experiments to support model reuse, as simulation experiments for validation are capable of representing information of model context and the corresponding desired model behavior, which is essential to effective model reuse. Section 3.3 was devoted to discussing related work: some other work that promotes the concept of reusing simulation experiments, and the regression testing in software development, which shares some similarities of the presented idea.

Afterwards, the requirements of reusing simulation experiments for model reuse were analyzed in Section 3.5.4. A central requirement is an explicit specification of simulation experiments. In the experiment specification, the experiment conditions, including model configuration and simulation set-up, are unambiguously described. Besides, to represent the validity information of the model, the expected behavioral properties the model exhibits in the experiment should be explicitly defined as well. In addition, the data processing methods needed for analyzing experiment results also need to be specified.

Lastly, Section 3.5 described the proposed approach, and outlined three key elements of the approach: a method for reusable simulation experiment specification, a mechanism to automatically generate experiments with the new model based on specifications of reused models, and a mechanism to automatically analyze results. Furthermore, the conceptual architecture of the approach was described, which consists of three layers, each corresponding to one key element. In the prototypical implementation, while SESSL was selected for experiment specification, the tool TAECS has been developed to adapt, execute, and check experiment specifications.

The presented approach will be applied to two types of model reuse, model composition in Chapter 4 and model extension in Chapter 5.

Chapter 4

Model Composition Based on Reusing Simulation Experiments

The previous chapter has illustrated how simulation experiments can be exploited to support model reuse and the general idea has been introduced. This chapter will look into a specific type of model reuse, i.e., model composition, and study the reuse of simulation experiments for developing composed models.

First of all, an introduction of model composition is presented in Section 4.1, and the processing of developing models by composition is discussed based on a life cycle in Section 4.2. Next, the approach of reusing simulation experiments for model reuse is studied in the context of model composition and problems that need to be considered for this type of model reuse are tackled (Section 4.3). Finally, to illustrate how the developed approach can be applied to support the development of composed models, two case studies are presented: the composition of Lotka-Volterra models (Section 4.4) and the development of a Wnt/β-catenin signaling pathway model by successive composition (Section 4.5).

4.1. Introduction on Model Composition

The more models are developed and available, the more reusing existing models as a basis to develop new ones becomes interesting. One common way to reuse models is model composition, where individual models are reused and composed together to create larger models. Models that are selected for composition may be developed for different purposes, in different contexts, and by different groups. In this case, models are developed for the intention of being reused by third parties in a "plug-and-play" fashion [51], which is often referred to by the modeling and simulation community when speaking of model composition. Alternatively, models can be developed and

reused within one group or community, i.e., firstly building small or simple models individually, and then composing them into a larger, more complex model. In this situation, models are developed in a "divide-and-conquer" manner, which facilitates the modeling of complex systems.

According to [158], models with the same level of abstraction can be composed to build models that cover larger modeling scope, i.e., horizontal composition; also, models with different levels of abstraction can be composed to capture the phenomenon in more detail, i.e., vertical composition. Thereby, building models by composing and integrating existing ones not only holds the promise of reducing the required time and effort for the model development, but also facilitates the development of more realistic, complex models to represent a wider range of phenomena.

4.1.1. Composition and Composability

Typically, model composition refers to building complex models from pre-existing "components". In the traditional component-based modeling, model components are viewed as portable building blocks [238], with interfaces defined to provide necessary information for reuse of components [53], and a composed model is created by connecting the inputs and outputs of components [21]. Developed independently as a replaceable part of a system, a model component should be stored in a library, and reused in unforeseen contexts and for different purposes [238]; a new model can be built by selecting needed components from the library and composing them together, i.e., the ideal of "plug-and-play" model development. In many technical areas (e.g., mechanical systems), libraries of such model components exist and have been proven highly effective for a rapid modeling of different systems [64].

In addition to the building-block model components (also termed as modules in some work, e.g., in [51]), other types of components are used for model composition as well. Depending on what is to be composed and what is the resulting composite, nine types of composition are distinguished in [188], with the component type varying from *Application* to *Behavior*.

In some non-technical areas, building-block components with clear and static interfaces are difficult to identify. For example, in the field of computational biology, although cells can be viewed as reactive systems with a clear boundary to their environment [73], which thus creates the impression that they can be treated as building blocks, cell biological modeling mainly focuses on intra-cellular dynamics and describing signaling or metabolic pathways as reaction networks [13]. Also, in those areas no libraries of building-block model components has been established yet; instead, public libraries of complete models can be found, such as the BioModels

Database [134] and the Physiome Model Repository 2 (PMR2) [269], which facilitates model exchange and often serve as starting points for model reuse. Thereby, complete models can also be reused for model composition. In this thesis, the term "model component" refers to not only model building blocks, but also complete models, each of which can be reused as part of another larger model.

Regardless of the type of model components (model building blocks or complete models) and the means of achieving the composition ("plug-and-play" or "divide-and-conquer"), the resulting composed model must be coherent and meaningful. Composability is defined as "the capability to select and assemble components in various combinations to satisfy specific user requirements meaningfully" [51]. Often composability is divided into different levels, such as the syntactic and semantic composability [15, 188]. Syntactic composability is concerned with component connections and their communication, i.e., whether the components can be connected compatibly with respect to the model definition and implementation such as parameter passing. In contrast, semantic composability focuses on whether the model components can be composed in a meaningful way and the composed model is valid. In addition to syntactic and semantic composability, another level of composability is distinguished by del Milagro Gutierrez and Leone [57], i.e., pragmatic composability, which focuses on whether the model components are aware of the simulation context. Similarly, Medjahed and Bouguettaya [158] identify four levels of composability: the syntactic, static semantic, dynamic semantic, and qualitative level. Whereas static semantic composability refers to the meaningful interactions between model components that are not related to the execution, dynamic semantic composability is concerned with the dynamical behavior of the composed model and qualitative composability assesses qualitative properties of the composed model with predefined quality metrics. Based on several efforts on composability and interoperability, the Levels of Conceptual Interoperability Model (LCIM) was developed, which comprises six levels, i.e., the technical, syntactic, semantic, pragmatic, dynamic and conceptual level [246]. Despite different ways to define composability levels, in general it can be summarized into two levels: the syntactic and semantic composability.

4.1.2. Existing Work to Support Model Composition

It has been widely recognized that composing models to form a new one is non-trivial and poses many challenges [51, 177]. Therefore, a great deal of effort has been put into facilitating model composition and coping with the diverse challenges.

Page and Opper [177] identified the complexity introduced by model composition, such as the possible challenges posed by composing models with different level of

abstraction, the difficulties in identifying suitable candidates from possibly massive component repositories, and the complexities of determining whether the composition satisfies modeling objectives. Based on a conceptual framework for composable simulations, Kasputis and Ng [106] from a system's perspective addressed the issues that should be considered to achieve composability and showed that such a framework is beneficial for model composition. Also, the authors outlined several challenges and needed research directions, and concluded that "unless models are designed to work together, they don't (at least not easily and cost effectively)". Petty and Weisel [188] discussed the lexicon of composability, including its definition and different composability levels. In the context of DoD (Department of Defense) applications, Davis and Anderson [51] identified the factors that affect composability and provided some suggestions to improve model composability. Moreover, Davis and Tolk [52] discussed how composability and multi-resolution modeling are related and how ontologies can be used to improve both.

In addition to the work with the focus on the conceptual and theoretical level as described above, a variety of approaches can be found in facilitating model development via composition from different aspects such as component selection [259] and composability check [148], and in different application domains such as systems biology [160] and electrical systems [58]. Also, several general-purpose frameworks exist to support model composition, such as the DEVS (Discrete Event System Specification) [272], the OSA (Open Simulation Architecture) [48], the BOM (Base Object Model) [166], and the CODES (COmposable Discrete-Event scalable Simulation) [242].

The diversity of approaches reflects the complexities and requirements of model composition. One way to structure those approaches and deal with the complexities is to examine the process of developing a model by composition. The M&S life cycle has been proposed to organize the traditional modeling process, e.g., [9, 215]. This can serve as a basis to structure the process of developing models via composition and identify crucial steps, i.e., the life cycle of developing composed models.

4.2. A Life Cycle of Developing Composed Models

Based on the focus of the study, the modeling process can be structured into different modeling and simulation life cycles, e.g., [6, 9]. In general, the process of traditional model development can be summarized into five stages, which include problem identification, conceptual model development and validation, simulation

model development, simulation model verification, and simulation model validation, as presented in the life cycle from Sargent [215]. When models are developed through composing existing ones, the model development process shows some differences from the traditional one, e.g., the simulation model development stage needs to be examined in more detail for the model composition process, such as dividing it into two stages: the selection of model components and the construction of the composed model.

Depending on how model composition is supported, the process of developing the composed model can be structured from different perspectives. For example, in [106], the presented conceptual composable simulation framework contains six main stages, i.e., identifying user requirements, translating user requirements into model requirements, selecting components, developing possible candidates for the composed model, selecting the best candidate, and system evaluations. In the framework for composing models proposed by Röhl et al. [203], the composition process is divided into four stages, i.e., instantiating model components, customizing model components, composing components, and transforming into simulation model. Szabo et al. [235] proposed a component-based model development life cycle, which consists of four main stages, i.e., conceptual model definition, syntactic verification, model discovery and selection, and semantic validation.

In this thesis, the process of model development via composition is structured into a life cycle of six phases, as depicted in Figure 4.1. This life cycle starts with the stage of building the conceptual model. Deriving a satisfactory conceptual model is a non-trivial task that may require multiple subtasks, which however is beyond the scope of this thesis and thus simplified into one phase. Based on the work presented in [9, 215], the first phase of the life cycle incorporates the *problem formulation, requirements engineering* and *conceptual modeling*, which is denoted as the *PRC* phase. The conceptual model is developed at the end of the PRC phase. Subsequently, the simulation model is constructed from components, after the two phases *model selection* and *model composition*. The two phases *syntactic verification* and *semantic validation* ensure that the composed model is correctly defined and meaningful. Once validated, the composed model can further serve as a starting point for building larger models, i.e., models can be developed through successive composition. To allow further reuse, documentation of the composed model is required, which is fulfilled in the phase *model documentation and storage*. Similar to all other modeling life cycles, although six phases are distinguished, the whole modeling process can be iterative.

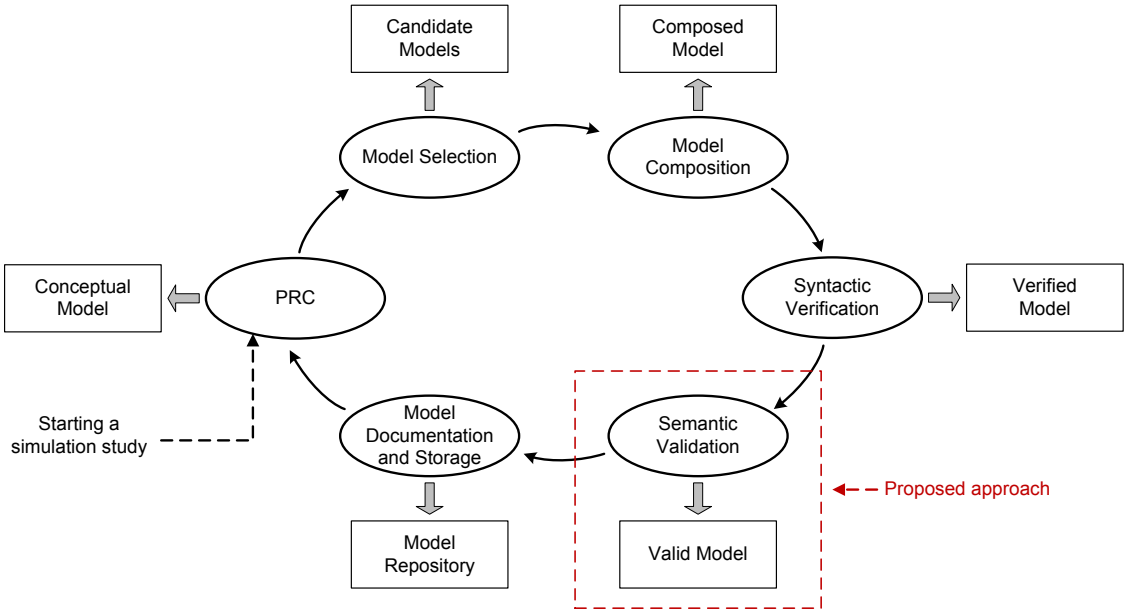


Figure 4.1.: A life cycle of modeling based on composition. PRC stands for Problem Formulation, Requirements Engineering and Conceptual Modeling, which are merged into one phase in this life cycle with a conceptual model developed. Usually a simulation study starts with the PRC phase.

Problem Formulation, Requirements Engineering and Conceptual Modeling

Developing the conceptual model often involves analyzing the problem entity and defining the objectives and requirements of the simulation study. A variety of work has been dedicated to the development of conceptual models; however, no general, wide agreement has been achieved on what is a conceptual model and what belongs to the conceptual modeling [33, 202]. A definition is given by Robinson [200] that the conceptual model is *a non-software-specific description of the computer simulation model (that will be, is or has been developed), describing the objectives, inputs, outputs, content, assumptions, and simplifications of the model.* Despite different views and terms used, it is widely agreed that the conceptual model is one of the first products in a simulation study and is independent from the simulation language or simulator [33]. To achieve effective model composition, the importance of the conceptual layer needs to be addressed, as indicated by the Levels of Conceptual Interoperability Model (LCIM) [246] (see Section 4.1.1), and efforts towards conceptual modeling are needed [247]. It has been pointed out by Tolk et al. [247] that the representation of conceptual models to support composition need to be formally specified in a machine-understandable language, and possible approaches include ontologies, Petri nets, and UML [33, 167, 242, 247].

Model Selection

When repositories of models are available, modelers can search the repository and select suitable components, based on the defined conceptual model. Typically, the selection of models for composition is realized based on matching between user queries and model repositories. Thus, the representation of user queries and models becomes essential. Often ontology and XML related techniques are exploited for representing model components, e.g., [167, 242]. Other work proposes to annotate models with additional information about the model. For example, to allow formalism-independent model retrieval and selection, model meta-information is provided using the Resource Description Framework (RDF) in [117], and to facilitate semantic-based model selection, an XML Model Description with semantics information represented in tag-based ontologies is proposed in [259]. Moreover, some mechanisms are developed to quantitatively measure similarity and rank model candidates, such as calculating a matching index regarding attributes and behavior [235].

Model Composition

When suitable model components are selected, they need to be composed together. Various methods exist to generate composed models from components, as discussed in [169]. One common way is the black-box composition, where the composed model is generated by connecting the inputs and outputs of model components [51, 106, 188, 242]. In this type of composition, model components are viewed as building blocks with predefined interfaces, and the internal structure and implementation of model components are hidden from the users. For many technical areas, this type of composition is widely used, as models for composition can be defined as such building blocks. For instance, a vehicle model can be generated by connecting interfaces of components such as vehicle framework and steering system [198].

In contrast, as discussed in Section 4.1.1, in some cases, representing models as portable building blocks is difficult to achieve, such as in computational biological modeling. Thus, the traditional black-box connection is not applicable to compose this type of models. Instead, a white-box integration method that merges variables, structures, and reactions of different models to be composed is required. Several approaches have been presented to compose such models, e.g., *fusion*, which merges model components into a single unified model without redundancies and the identities of the original models are lost, and *composition*, which describes the "glue" that holds the models together, as described in [194, 195]. Similarly, for reaction-based models with hierarchically nested structures in cell biology area, model composition

in a white-box manner is needed [183]. Additionally, other types of composition are identified, such as glass-box, where the internal structure of model components can be accessed but not modified [232, p. 5].

Syntactic Verification

Independent of whether the composed model is built from model building blocks or from complete models, through connection of model inputs and outputs or through merging of model internal structures, the resulting composition needs to be consistent and coherent. First of all, the consistency on the syntactic level is required, i.e., syntactic verification. In this phase, the syntactic composability of model components needs to be checked to ensure the components can be connected correctly.

In the traditional model composition that is realized by connecting model building blocks, syntactic verification is mainly concerned with the interaction between model components, e.g., whether the data exchanging is compatible. In this case, checking syntactic composability is often based on the model representation provided by the composition framework, e.g., in [203] model components and their composition are represented in XML and syntactic verification relies on syntax checking of XML documents, and in [233] model components and their composition are specified using regular grammars and syntax checking is realized through predefined composition grammar rules. When components are composed in a white-box manner, the internal structures of components are merged. In this case, syntactic verification involves the consistency of the whole model definition, e.g., matching variables and compartments of different models. Possible conflicts are typically checked by the user, and some tools have been developed to provide additional assistance. For example, the semanticSBML [120] and SBMLmerge [218] can assist in identifying biologically equivalent reactions and species among the models to be merged, by making use of SBML annotations. JigCell [252], which supports different approaches to compose models, i.e., fusion, composition, aggregation and flattening [194, 195], allows users to define models as spreadsheets so that models can be fused and composed via "mapping tables", or models can be defined as graphical box diagrams and then be aggregated by linking input and output ports. In [44], a web-based tool is presented to compose SBML models, which supports merging models by a semi-automatic algorithm based on matching names and annotations.

Semantic Validation

Syntactic verification only ensures a consistent composition at the syntax level, i.e., the definition of the composed model. To obtain a meaningful composition, the

behavior of the composed model is of significant importance for the overall coherency as well, i.e., the validity of the composed model.

However, the semantic validation of the composed model remains a grand challenge in modeling and simulation [236]. First of all, valid model components do not guarantee a valid composition. Secondly, model components may be developed for different purposes and in different contexts, which could lead to conflicts during the composition and emergent properties may arise in the composed model. Also, the context in which the composed model is developed might be different from the components, and thus the behavior of the composed model may diverge from its components, e.g., certain behavior of model components is expected to be violated in the composed model. The problem is further complicated when complete models are reused as components. In contrast to model building blocks that are designed to be used in "unforeseen" context [238], complete models are created to answer certain questions under specific contexts, and therefore are not able to be easily composed with other models in an arbitrary context [51, 175]. Thus, in order to check whether the components are suitable for the composition in the semantic level, comparing the behavior of the composed model and its components is necessary. Last but not least, the composed model needs to be valid with respect to the modeled system and the according context.

Although a great deal of effort has been devoted to facilitating model composition, research on semantic validation of the composed model is limited and still under development. A formal theory for semantic composability is presented in [189, 262]. By defining models as mathematical computable functions, and simulation as the sequential execution of a model and represented by an Labeled Transition System (LTS), model composition can be viewed as compositions of mathematical functions. Thus, semantic composability is evaluated by comparing the simulation of the composed model and the simulation of a perfect model based on LTS theory. Szabo and Teo [234] propose a three-layered approach to validate the composed model. Other work on semantic composability exist, e.g., formal method based on Z specification for DEVS models [249] and semantic information comparison and state machine execution for composition of BOMs [167].

Model Documentation and Storage

Model documentation and storage has increasingly attracted attention, as suitable documentation and accessible storage is crucial to facilitate model exchange and reuse [175]. Also, the documentation and storage of models has a significant impact on other phases in the life cycle, such as model selection.

Some work has been presented aiming at the design of reusable models. In order to improve the model reusability, several guidelines have been presented by Verbraeck and Valentin [253] to design simulation building blocks, involving different aspects such as self-containment and interoperability. In the project SPICOSA (Science and Policy Integration for COastal System Assessment), efforts have been made toward designing high-quality reusable model components to allow developing new models for different contexts, with component design guidelines proposed [55]. Besides, some work focuses on defining models in a manner that supports model composition, such as the definition of coupled models in DEVS and the specification of hierarchical model definition in SBML [229].

However, efforts only on the design and representation of models may not be sufficient for effective reuse, e.g., models are usually represented in a specific modeling formalism or language, which could be ambitious or difficult to understand for third-party users. Thus, additional information about the model is necessary, such as the context in which the model is developed and the data used for the model development. The Minimum Information Requested In the Annotation of biochemical Models (MIRIAM) was proposed to define a standard for encoding models of biological systems [171], and contains two parts, i.e., a standard for reference correspondence to provide information on the syntax and semantics of the model, and an annotation scheme for specifying the attribution annotation and external data resources. The ODD (Overview, Design concepts, and Details) protocol has been introduced as a standard format for describing individual-based and agent-based models with the objective of facilitating understandable and complete model descriptions and to address the problem of reproducibility and reuse [82, 83]. Ontologies are often used to annotate models with semantic information, and some tools have been developed to annotate and compose models based on ontologies such as SemGen [76].

Meanwhile, as the number of developed models increases, so does the number of model repositories, e.g., the BioModels database [134], the CD++ repository [32] and the CoMSES Net Computational Model Library [204]. With the size of models and repositories growing, model retrieval and selection might face certain challenges, e.g., how to efficiently identify and retrieve suitable candidates, and approaches on the design and organization of model repositories are needed, e.g., as presented in [96, 141, 259].

4.3. Reusing Simulation Experiment for Model Composition

In general, most of the current work on model composition aims at properly defining a composed model and focuses on specific steps in the life cycle, such as composing model candidates and checking the composition at the syntactic level. As discussed in Section 4.2 (pp. 78-79), checking semantic composability of models is a challenging endeavor with limited work existing, this thesis aims to support the semantic validation of the life cycle, as shown in Figure 4.1.

Existing approaches on semantic composability typically focus on situations where model components are connected via inputs and outputs. For instance, approaches presented in [234, 262] are based on viewing models as commutable functions with inputs and outputs, and thus cannot be used for cases where identifying model inputs/outputs is difficult and models are composed in a white-box merging manner. Besides, in those approaches, the validation of the composed model relies on analyzing the execution sequence of the composed model and comparing it to a reference (a perfect model [237] or a pre-specified scenario [167]), either by transforming model executions into Labeled Transition Systems [237, 262] or based on state-machine execution [167].

This thesis takes another perspective and employs simulation experiments that are performed with individual models to check the model validity. In the previous chapter, the approach has been presented, which exploits simulation experiments of the reused models to automatically generate experiments for the new model, so that information on the validity of the model reuse is provided. With the general concept described in Section 3.5, in the following the proposed approach will be discussed and applied in the context of model composition.

4.3.1. Overview

An overview of reusing simulation experiments to support model composition is given in Figure 4.2. As presented in the previous chapter, the approach relies on explicit specifications of simulation experiments. First of all, models are annotated with the specifications of simulation experiments that have been executed for validation, corresponding to step 1 in Figure 4.2. In the specification of experiments, the desired model behavioral properties that indicate the validity of this model are explicitly defined, as well as the experiment conditions to generate this key behavior, including model configuration and simulation configuration. Afterwards, as shown in step

2, individual models are reused for model composition. Based on how models are composed, the user can optionally specify information for experiment adaptation when necessary, such as renaming and reassignment of model parameters. Besides, result expectations can be defined by the user to indicate, when checked on the composed model, which properties of the individual reused models are expected to hold and which not. Subsequently, the annotated experiment specifications of individual models are reused and adapted for the composed model, corresponding to step 3. Afterwards, the adapted experiment specifications are executed with the composed model and the experiment results are checked, to test whether the behavioral properties, which are defined in the experiment specifications of the reused models and have been successfully checked, still hold for the composed model, and whether this result meets the expectation, corresponding to step 4. The developed tool TAECS is used to automatically adapt and execute experiment specification, and check experiment results against properties, corresponding to the working case A (Figure 3.9) and the Algorithm 1 (Section 3.5.3, p. 63). The main goal is to gain insights into semantic composability of models by reusing their validation experiments.

As illustrated in Section 3.5 (p. 56 and 66), three main elements are needed for this approach: reusable specifications of simulation experiments, experiment generation from specifications, and result analysis based on explicit property specification and checking algorithms. While experiment specification relies on SESSL and property specification depends on characteristics of the modeled system, generating simulation experiments based on experiment specifications of reused models needs to take into account the type of model reuse, as discussed in Section 3.5.3 (p. 64). The next section will be dedicated to experiment generation for the composed model by reusing experiment specifications of model components.

4.3.2. Experiment Adaptation and Generation

As discussed in Section 3.5.3, while generating simulation experiments for the composed model based on experiment specifications of model components, those reused experiment specifications may require adaptation.

In addition to an update in model location (i.e., from the model component to the composed model) and possible parameter renaming, special attention needs to be paid to the adaptation of model parameter configuration, which results from the fact that the composed model contains all the parameters from model components and model components may share some parameters. Thus, model parameter assignments in each experiment specification need to be refined in order to suit the composed

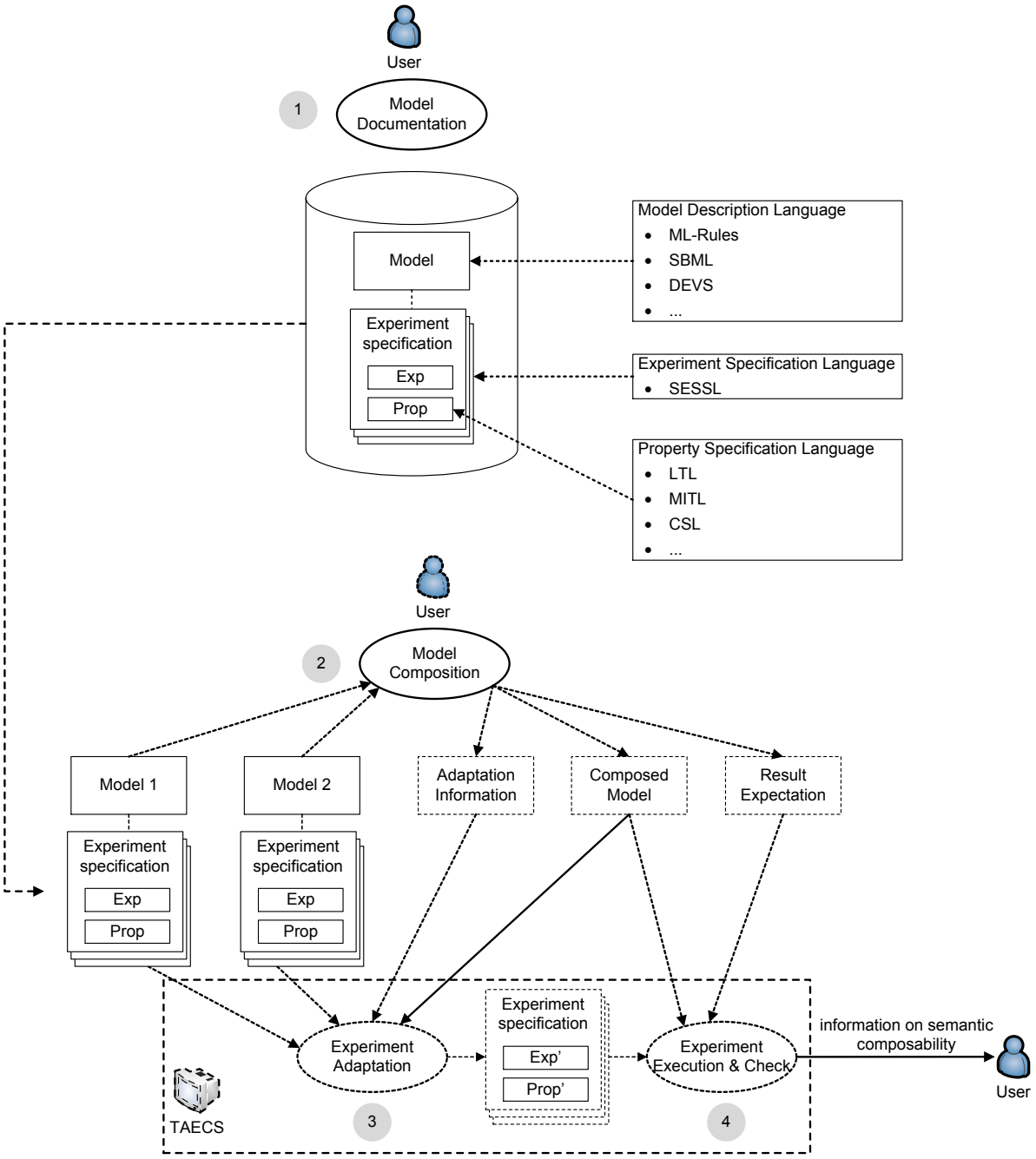


Figure 4.2.: The overview of reusing simulation experiments for model composition. Models are annotated with explicit specifications of the simulation experiment executed for validation, and each experiment specification contains the expected behavioral property and the experiment conditions used to produce the property. When two models are composed, their experiment specifications are reused and adapted to automatically generate experiments for the composed model. The behavioral properties that hold for the individual reused model are checked against the experiment results to inspect whether the composed model exhibits the behavior or not and whether this is in agreement with user expectations, therefore to provide insights on the validity of the composition.

model. A specific strategy is required to determine the configuration of those model parameters that only exist in the other model and those that exist in both models but are configured differently. This is handled in the function `updateParameterConfig`, which is called by Algorithm 1 (Section 3.5.3, p. 63).

Different ways can be used to adapt model parameter configuration in the experiment specification of model components in order to generate experiments for the composed model. One solution is to define a set of default configurations for all model parameters in the composed model file, and for those parameters which should be limited to a specific configuration, their configuration is defined as part of the adaptation information. When those parameters are explicitly specified in an experiment specification of model components, their configuration will be updated to be the one contained in the adaptation information. In this case, parameter configuration in the composed model and possible conflict resolution completely rely on the user.

Some algorithms can be employed to provide assistance in adapting model parameter configuration. Currently, one method is implemented in the prototype, which generates a possible assignment range for each parameter that appears in experiment specifications of model components. First of all, for each model component, its associated experiment specifications are iterated over to obtain all model parameters that are explicitly configured in those experiment specifications. For each specified model parameter, the minimum and maximum value of its configuration in all experiment specifications are calculated, respectively. A boundary is constructed using this minimal and maximal value, which constitute a possible assignment range for this parameter. For model parameters that are not contained in any experiment specification of the model component, their configuration must have been given in the model file and it is likely that those parameters should be limited to this configuration for the purpose this model component is built for. Therefore, for those model parameters, the configuration in the model file is used to execute all experiment specifications. If a parameter is defined in both model components, the overall minimum and maximum value of its configuration in the experiment specifications from two components, are used to construct the parameter range. In this way, possible valid parameter ranges for the composed model can be defined without the users' intervention. When new model parameters are introduced in the composed model, which do not exist in any model components, their configuration should be provided either in the model file or in the adaptation information. Still, the modeler can explicitly set one parameter to a constant value during the composition, defined as part of adaptation information, which will update the assignment range

of this parameter from the one calculated based on experiment specifications to the constant value. Further, for each parameter in the experiment specification to be adapted, the intersection between the configuration in the experiment specification and the obtained assignment range is calculated and used.

This method is implemented as a proof of concept, and there are other ways to generate possible assignments, e.g., for each parameter that is configured in experiment specifications, an assignment range can be constructed by computing an intersection of all intervals specified in experiment specifications, in which case, the calculated assignment range could be empty.

Therefore, after the adaptation described above, the value of model parameters may be assigned to a range, which could be large. To perform experiments efficiently, sampling over the range is used to generate parameter assignments. As a proof of concept, the current implementation of the prototype samples uniformly from the given range; other sampling methods can also be used, such as orthogonal latin hypercubes (e.g., [212]). A default sampling size of 10 is configured in the implementation. It is expected that the sampling size and the quality of the sampling method will have a large impact on the overall performance. Users can also adjust the sampling procedure to their requirements, e.g., to generate more samples if the available hardware is sufficiently powerful. This can be expressed via the user preferences. Similarly, more sophisticated sampling methods may require additional information on the actual model composition, e.g., to generate more samples for parameters shared by both model components.

Using the sampled assignments, the simulation experiment is executed with the composed model and results are checked against the specified property. If the property checking fails for the experiment, it will return a (set of) counterexample(s), i.e., the parameter configuration with which the experiment result does not satisfy the defined property. It is easy to re-check these counterexamples on the model for which they have been originally defined. This is important, because the experiment may have been checked insufficiently for the model components, i.e., the property may not even hold for the original models with those parameter configurations. Depending on the outcome of testing the corresponding model component, the user can find out whether there is a problem with the composition (i.e., the counterexample is true for model component, but not for the composed model) or a problem with the original model component (i.e., the counterexample is true for neither model components nor the composed model).

The two methods for model parameter adaptation discussed above, i.e., the method relying on parameter configuration in the model file and adaptation information

provided by the user, and the method based on automatically calculating parameter assignment ranges, are applicable for different scenarios. When the assignment of model parameters is limited to individual values (especially only a few variants) instead of a range, the first method is more suitable and a sampling of the range constructed from those values could offer little help, as only those individual assignments are of interest. However, when model parameters are configured as value ranges, the second method is more helpful and able to provide more information. In the prototype implementation, users can choose which method to use.

4.4. Case Study I: the Lotka-Volterra Model

In this case study, the proposed approach has been applied to the composition of Lotka-Volterra models and the main purpose is to demonstrate how to use the proposed approach for semantic validation of the composed model.

4.4.1. Background

The Lotka-Volterra model, described in [196] and attributed to [140, 254], is one of the classic predator-prey models and also a well-known example of a mathematical model of population biology. It describes the interaction between predators and prey in an abstract manner. In this case study, a specific variant of the Lotka-Volterra model was used, where one predator species hunts one prey species and the prey depends on an unlimited supply of food. To avoid an exponential growth of the prey population in the absence of predators, competition among prey was considered. Let N_1 denote the prey population size and N_2 denote the predator population size, and the deterministic equations are as follows (cf. [196, p. 176]):

$$\mathrm{d}N_1/\mathrm{d}t = N_1 * (b - k * N_1 - a * N_2) \qquad (prey)$$
$$\mathrm{d}N_2/\mathrm{d}t = N_2 * (-d + c * N_1) \qquad (predator)$$

where b is the birth rate of the prey, d is the mortality rate of the predator, a and c are the interaction coefficients, and k is the competition coefficient.

Different questions can drive the modeling and simulation of prey and predator systems. For example, one may be interested in the stationary states of the system, e.g., the “coexisting state”, where both prey and predator populations exist, the “prey state”, where the predators die out and only prey survive, or the “empty state”, where predators die out after the prey population is extinguished [61]. Similarly, it may also be interesting to compare the difference in the predator and prey populations, e.g., the “recovery comparison”, which states that the prey population will recover faster

than the predator population if both populations have been disturbed significantly, i.e., both populations are quite small. These questions can be reflected in simulation experiments with the model, i.e., under which circumstance (such as parameter values and initial state), the model behavior satisfies corresponding properties. For demonstration, three properties were chosen in this case study, i.e., the "coexisting state", the "empty state", and the "recovery comparison".

When two basic Lotka-Volterra models are composed together, certain questions arise: what behavioral properties will show in the composed model? will those properties that have been checked successfully in the two model components still be preserved in the composed model? What insight does a violation tell us about the semantic validity of this composition? In the following, the developed approach, i.e., reusing simulation experiments of model components to automatically generate experiments with the composed model, is used to provide assistance in answering those questions.

4.4.2. Ingredients

First of all, a modeling formalism and the corresponding simulation algorithm are needed for performing simulation experiments. As discussed in Section 3.5.5, to apply the proposed approach for model composition, three elements are required: a method for experiment specification, a mechanism for experiment adaptation and generation, and a mechanism for automatic result analysis. SESSL is used for experiment specification throughout this work. For illustration, the method based on calculating parameter assignment ranges has been used for experiment adaptation and generation in this case study. The M&S method and result analysis will be described as follows.

Modeling and Simulation Method

ML-Rules was selected to describe the Lotka-Volterra models, which is a rule-based, multi-level language for modeling cell biological systems [94, 153, 261]. It allows the description of dynamic hierarchical structure with nested species and the assignment of attributes and solutions to species at each hierarchy level. Also, it supports downward and upward causation across different levels of the hierarchy. To facilitate a concise description of complex models, rule schemata are used in ML-Rules. Besides, for each rule, a flexible definition of reaction rate kinetics using any kind of mathematical expression and constraints is allowed.

For the execution of ML-Rules models, the reference stochastic simulation algorithm from [153] was used, which is based on Gillespie's approach [78], and referred as the case class named `MLRulesReference` in SESSL. The modeling and simulation framework JAMES II was exploited as the simulation system to actually execute the experiments specified in SESSL.

Property Specification and Checking Algorithms

As discussed in Section 3.4.3 (pp 51-54), various approaches can be employed for specifying model behavioral properties. Linear Temporal Logic (LTL) is widely used to check the properties of individual trajectories [70, 274], and allows to express a broad range of dynamic model properties. For two of the interesting behavioral properties in this case study, the "coexisting state" and "empty state", LTL can be used for property specification. To check an individual trajectory against a property specified in LTL, the model-checking algorithm introduced by Fages et al. [70] was exploited and reimplemented as part of JAMES II. The original algorithm only covers the following operators of LTL: X (next), G (global), F (finally), and U (until), and was extended to support the R operator (release) (see [36]).

Since ML-Rules is based on Continuous-Time Markov Chain (CTMC) semantics, the simulation results of ML-Rules models are stochastic and generated simulation trajectories contain stochastic noise. Therefore, LTL might not suffice to describe the third property of interest, i.e., the "recovery comparison", which involves the expression of "increase" with noise considered (see Section 3.4.3, pp. 52-53). As discussed in Section 3.5.4 (p. 65), due to the extensibility of SESSL, the presented approach is open to new property specification methods. Thus, a custom predicate was defined to express the "recovery comparison" property, with the corresponding algorithm provided to check the trajectory.

For experiments with stochastic models, multiple replications are required for analysis. Due to the stochasticity, with the same simulation set-up, a property may hold for some replications and not for others. Thus, the expression of probability regarding replications is necessary. The Continuous Stochastic Logic (CSL) [222, 268], which has been proposed as a formalism for expressing properties of CTMC, was exploited to support the definition of probabilistic statements as $Pr_{\bowtie p}(\phi)$, where $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0,1]$, and ϕ could be an LTL formula or a predefined predicate. In contrast to [222, 268], however, currently the prototypical implementation does not support nested probabilities.

Statistical model checking techniques are frequently applied to determine whether a stochastic model satisfies a specification in the form $Pr_{\bowtie p}(\phi)$, e.g., $Pr_{\geq p}(\phi)$ [128]

(see discussion in Section 3.5.4, p. 65). Based on checking sampled traces of the state space, i.e., the individual simulation trajectories, such techniques can decide whether the probability p' that a randomly selected simulation run satisfies the property ϕ is not smaller than a threshold p, and *hypothesis testing* is employed to make this decision [266].

Whether $p' = Pr(\phi)$ is at least p can be expressed with two competing hypotheses: the null hypothesis $H_0 : p' < p$ states that $Pr_{\geq p}(\phi)$ does not hold; the alternative hypothesis $H_1 : p' \geq p$ states that $Pr_{\geq p}(\phi)$ holds. By rejecting H_0, some evidence can be gained in H_1,i.e., $Pr_{\geq p}(\phi)$ holds; if it fails to reject H_0, some evidence can be gained against $Pr_{\geq p}(\phi)$. Hypothesis testing cannot guarantee that the result is correct; however, one can improve the confidence of testing by limiting the probabilities of type I error (rejecting H_0 although it is true, false positive) and type II error (failing to reject H_0 although it is false, false negative), termed as α and β, respectively. Smaller values for α and β correspond to more confidence in the test result; however, a low probability for both error types is difficult to achieve (see [267] for detail). To make a small value for both α and β possible, typically an indifference region of width δ around p is introduced. Thus, the hypotheses $H_0 : p' < p - \delta$ and $H_1 : p' \geq p + \delta$ are used instead.

Various methods exist to determine which hypothesis to accept, such as acceptance sampling with fixed-size samples and sequential acceptance sampling [267]. The approach presented by Sen et al. [223] has been adopted and implemented in the prototypical implementation of this work: the hypothesis H_0 is rejected when after executing n simulation runs, more than $p \times n$ runs satisfy ϕ. The minimal number of needed replications n can be found based on given p, δ, α, and β, by increasing n until the type I error probability $a \leq \alpha$ and the type II error probability $b \leq \beta$, as depicted in Algorithm 2. The error probabilities can be computed by assuming real probabilities $p' = p \pm \delta$ for given p, n, and δ [223]. Currently, in the prototypical implementation TAECS, α, β, δ are all configured with default value 0.05; however, they can be specified by the user as well.

4.4.3. The basic Lotka-Volterra Model

Based on the description in Section 4.4.1, the basic Lotka-Volterra model can be expressed in ML-Rules as shown in Figure 4.3, which depicts the definition of model parameters, species, initial state, and reaction rules. In ML-Rules, predators and prey are modeled as populations whose individuals encounter the events of death or reproduction in a stochastic manner. In the original Lotka-Volterra model equations, seven parameters are distinguished, i.e., a, b, c, d and k, as well as the initial predator

Algorithm 2 Algorithm for determining the number of replications needed to check a property. The number X of simulation runs satisfying the temporal logic formula is binomially distributed, with F being the cumulative distribution function.

```
calculateReplicationNumber(p, δ, α, β)
   n ← 1
   sufficient ← False
   while(not sufficient)
      n ← n + 1
      a ← P(X > p × n | X ∼ Bin(n, p − δ)) = 1 − F(n × p; n, p − δ)
      b ← P(X ≤ p × n | X ∼ Bin(n, p + δ)) = F(n × p; n, p + δ)
      sufficient ← a ≤ α ∧ b ≤ β
   return n
```

and prey population sizes (see Section 4.4.1), and the value of all those parameters can be set by modelers. For simplicity, in the model used for this case study, no distinction was made between the interaction coefficients so that both have the same value ($a = c$) and the competition coefficient was set to constant $k = 0.002$.

With the model defined, simulation experiments can be performed. A model with foxes being the predator and rabbits being the prey should illuminate the experiments. As stated above, three properties are of interest. The property "coexisting state", which states both prey and predator populations exist, can be expressed in LTL as:

$$G(\#Rabbit > 0 \wedge \#Fox > 0)$$

Assume that for a randomly selected simulation run, the probability that both predators and prey will survive is larger than 0.8, and this can be expressed with the combination of CSL and LTL as:

$$Pr_{\geq 0.8}(G(\#Rabbit > 0 \wedge \#Fox > 0))$$

The expression of probability property based on CSL (denoted as $Pr_{\bowtie p}(\phi)$) has been integrated into SESSL, and the above property can be expressed as:

```
assume{ (Probability >= 0.8)(
  G(variable("Rabbit") > 0 and variable("Fox") > 0)
  )}
```

For simplicity, the probability p was set to 0.8 throughout this case study. To make the model's behavior satisfy this property, the initial values of prey ($nRabbit$) and predators ($nFox$) were set to 100 and 10, respectively, and the rest of the parameters were fitted through parameter scanning against desired properties. The modeling and simulation framework JAMES II was employed for experiment execution and the simulator `MLRulesReference` was used. Each run stops after 10 simulation time units or after wall clock time 2 minutes. The configuration of model parameters for

```
a: 0.014;
b: 0.6;
d: 0.7;
k: 0.002;
nFood:100;
nPredator:10;
nPrey:100;
Food();
Predator();
Prey();
>>INIT[(nFood) Food +
      (nPredator) Predator +
       (nPrey) Prey];
// Prey reproduces
Food:f + Prey:x -> Food + 2 Prey @b*#x;
// Prey dies by competition
Prey:x + Prey:z -> Prey @k*#z*#x;
// Predator reproduces based on successful hunting
Predator:y + Prey:x -> 2 Predator @a*#y*#x;
// Predator dies
Predator:y -> @d*#y;
```

Figure 4.3.: A Lotka-Volterra model described in ML-Rules. [From [184]]

which the simulation output satisfies the "coexisting state" property is shown as follows:

$$a : 0.010 - 0.015, b : 0.6 - 0.7, d : 0.5 - 1.0$$
$$nRabbit = 100, nFox = 10$$

During the simulation, the number of rabbits (prey) and foxes (predator) were observed. An example of simulation results is shown in Figure 4.4, where model parameter configuration is $a = 0.01, b = 0.6, d = 0.5, nRabbit = 100, nFox = 10$. The experiment is specified in SESSL, as depicted in Figure 4.5.

Similarly, for the properties "empty state" and "recovery comparison", the same simulation experiment set-up was used and parameters were fitted to produce desired model behavior. For the "empty state" (i.e., the prey dies out first, then the predators die out), the property ϕ can be expressed in LTL as:

$$((\#Rabbit = 0)\ R\ (\#Fox > 0)) \wedge F\ (\#Fox = 0).$$

Taking the probability into account, the property is denoted as $Pr_{\geq 0.8}(\phi)$, and can be expressed SESSL as:

```
assume{ (Probability >= 0.8)(
  (variable("Rabbit") == 0) R (variable("Fox") > 0) and
      F(variable("Fox") == 0))
  )}
```

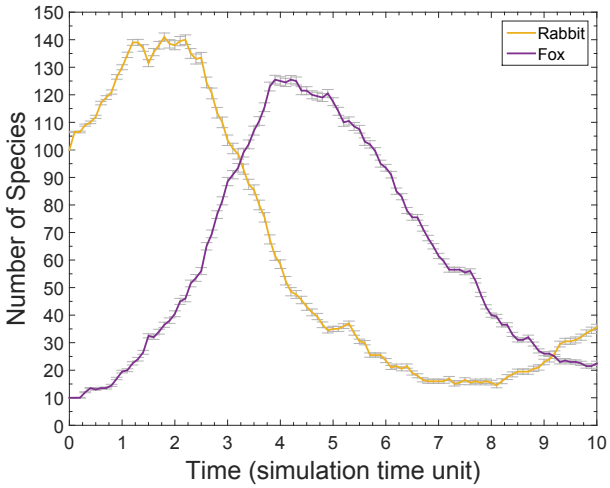


Figure 4.4.: Simulation results of the basic Lotka-Volterra model of the prey rabbit and predator fox regarding the "coexisting state" property, where model parameter configuration is $a = 0.010, b = 0.6, d = 0.5, nRabbit = 100, nFox = 10$. The mean trajectories with the standard error (in gray error bars) are shown.

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with Hypotheses {
  model = "file-mlrj:/./LotkaVolteraFoxRabbit.mlrj"
  scan( "a" <~ range(0.010, 0.001, 0.015), "b" <~ range(0.6, 0.1, 0.7),
        "d" <~ range(0.1, 0.1, 1.0))
  set( "nRabbit" <~ 100, "nFox" <~ 10)
  simulator = MLRuleReference()
  stopCondition = AfterWallClockTime(minutes=2) or AfterSimTime(10)
  observe("Rabbit","Fox")
  observeAt(range(0.0, 0.1, 10))
  assume{ (Probability >= 0.8)(
   G(variable("Rabbit") > 0 and variable("Fox") > 0)
   )}
}
```

Figure 4.5.: SESSL experiment specification regarding the "coexisting state" property for a basic Lotka-Volterra model of the prey rabbit and the predator fox.

The configuration of model parameters for which the simulation output fulfills the "empty state" property is shown as follows:

$$a : 0.050 - 0.070, b : 0.1 - 0.4, d : 0.7 - 1.0$$
$$nRabbit = 100, nFox = 30$$

An example of simulation results is shown in Figure 4.6, where the model parameter configuration is $a = 0.050, b = 0.1, d = 0.7, nRabbit = 100, nFox = 30$. The experiment is specified in SESSL, as depicted in Figure 4.7.

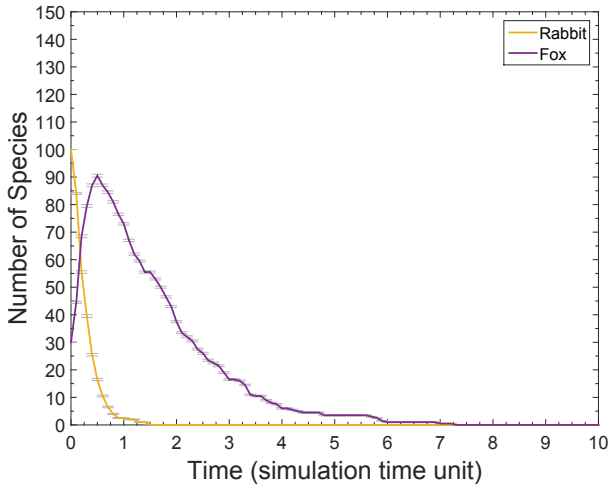


Figure 4.6.: Simulation results of the basic Lotka-Volterra model of the prey rabbit and predator fox regarding the "empty state" property, where the model parameter configuration is $a = 0.050, b = 0.1, d = 0.7, nRabbit = 100, nFox = 30$. The mean trajectories with the standard error (in gray error bars) are shown.

```
1  import sessl._
2  import sessl.james._
3  val exp = new Experiment with Observation with Hypotheses {
4    model = "file-mlrj:/./LotkaVolteraFoxRabbit.mlrj"
5    scan( "a" <~ range(0.050, 0.001, 0.070), "b" <~ range(0.1, 0.1, 0.4),
6         "d" <~ range(0.7, 0.1, 1.0))
7    set( "nRabbit" <~ 100, "nFox" <~ 30)
8    simulator = MLRuleReference()
9    stopCondition = AfterWallClockTime(minutes=2) or AfterSimTime(10)
10   observe("Rabbit","Fox")
11   observeAt(range(0.0, 0.1, 10))
12   assume{ (Probability >= 0.8)(
13      (variable("Rabbit") == 0) R (variable("Fox") > 0) and
14          F(variable("Fox") == 0))
15    )}
16 }
```

Figure 4.7.: SESSL experiment specification regarding the "empty state" property for a basic Lotka-Volterra model of the prey rabbit and the predator fox.

The "recovery comparison" property states that if both populations are disturbed at the beginning to have the same small size, the prey population will recover faster. This property involves the expression of increasing and comparison of the increasing rate for simulation trajectories with stochastic noise, which is difficult to specify in LTL. Therefore, this property is expressed with a predefined predicate (see Section 4.4.2). Detecting the increasing trend of trajectories with noise is a non-trivial task; quantitatively measuring and comparing increasing rate is even more challenging. One way to evaluate this predicate is to compare the time points where the first peak occurs in each population. The earlier the first peak occurs, the faster the population has recovered. Many sophisticated methods have been developed to find the optima in time series with noise (e.g., [62, 217]). For the purpose of illustration, a simple approach has been used: a certain time point is selected for checking and only observed time points between the initial and the selected time point are considered. Among those time points, the one with the maximum species number is identified, and the recovery rate is calculated as the slope between the initial value and this maximum. If the initial point has the maximum value, which means the species number does not increase within this time period, then the recovery rate is calculated as the slope between initial and selected time point. The number of selected observation points should not be too large, to avoid the case where species populations have already finished the recovery phase and may start to decrease. This approach may not be rigorous but appears to be sufficient for the purpose. In this case study, the fifth observed time point is selected for checking, which means only the first 5 observation points are considered for comparing the recovery rate of species. Although such a manual implementation of predicates is not too difficult in JAMES II, it is not acceptable for all users. Therefore, a language, which is able to specify properties with noise tolerance, would be preferable.

The predefined predicate for the property "recovery comparison" is named as `RecoveryComparison`. The initial number of both the prey and predator were set to 10, and the rest of the parameters were fitted so that the model behavior satisfies the "recovery comparison" property, as shown in the following:

$$a : 0.010 - 0.028, b : 0.6 - 1.0, d : 0.5 - 1.0$$
$$nRabbit : 10, nFox : 10$$

An example of simulation results is shown in Figure 4.8, where the model parameter configuration is $a = 0.010, b = 0.6, d = 0.5, nRabbit = 10, nFox = 10$. The experiment is specified in SESSL, as depicted in Figure 4.9.

All the three experiment specifications, i.e., Figure 4.5, Figure 4.7, and Figure 4.9, were tested on the model of rabbits and foxes (Figure 4.3), corresponding to the

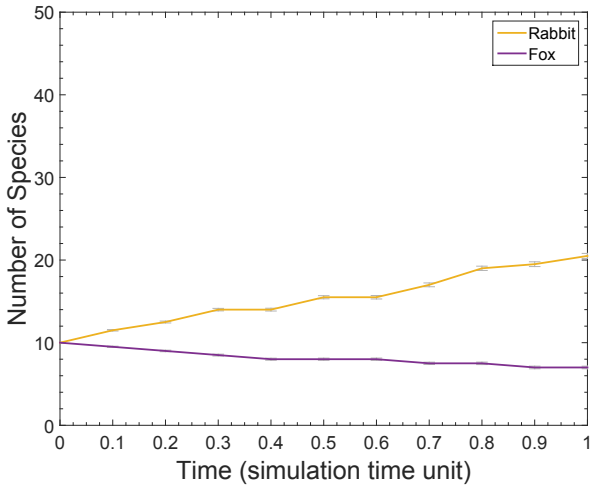


Figure 4.8.: Simulation results of the basic Lotka-Volterra model of the prey rabbit and predator fox regarding the "recovery comparison" property, where the model parameter configuration is $a = 0.010, b = 0.6, d = 0.5, nRabbit = 10, nFox = 10$. The mean trajectories with the standard error (in gray error bars) are shown.

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with Hypotheses {
  model = "file-mlrj:/./LotkaVolteraFoxRabbit.mlrj"
  scan( "a" <~ range(0.010, 0.001, 0.028), "b" <~ range(0.6, 0.1, 1.0),
        "d" <~ range(0.5, 0.1, 1.0))
  set( "nRabbit" <~ 10, "nFox" <~ 10)
  simulator = MLRuleReference()
  stopCondition = AfterWallClockTime(minutes=2) or AfterSimTime(10)
  observe("Rabbit","Fox")
  observeAt(range(0.0, 0.1, 10))
  assume{ (Probability >= 0.8)(
     RecoveryComparison("Rabbit","Fox")
  )}}
```

Figure 4.9.: SESSL experiment specification regarding the "recovery comparison" property for a basic Lotka-Volterra model of the prey rabbit and the predator fox.

case B (Figure 3.9). For all experiments, the method described in Section 4.4.2 was used to check whether for a randomly selected simulation run, a property holds with probability being no less than 0.8, and the determination of replication number relies on the Algorithm 2. With $\alpha = \beta = \delta = 0.05$ and $p = 0.8$, the calculated necessary replication number is 181. The three experiment specifications have been checked

successfully, and therefore the basic Lotka-Volterra model with rabbits as the prey and foxes as the predator was annotated with those experiment specifications.

4.4.4. Composing Lotka-Volterra Models

Given two basic Lotka-Volterra models, each of which has one prey and one predator, as shown in Figure 4.3, it is possible to compose them in several ways.

For example, two model components, which have the same prey but different predators, can be composed in a way so that in the composed model all three species exist and the two predators from both model components hunt on the same prey, namely the "same prey" composition. Similarly, when two model components have the same predator but different preys, the composed model could contain three species and the predator hunts on both prey species, namely the "same predator" composition. Besides, two model components may share a common species, which has different roles in the two models: as the prey in one model but predator in another model. The two components can be composed so that a food chain is formed in the composed model, where the shared species functions as both prey and predator, namely the "food chain" composition.

To illustrate the three types of composition, four basic Lotka-Volterra models have been created. In addition to the model with foxes and rabbits, as described in the previous section, three other models were defined, each having one predator and one prey, which are wolves and rabbits, wolves and foxes, and wolves and sheep, respectively. The model definition of these models are similar to the model in Figure 4.3, except for the names of the prey and predator species. Like the model with foxes and rabbits, each of the other three models has been annotated with three experiment specifications, regarding the three properties "coexisting state", "empty state", and "recovery comparison", as shown in Figures 4.5, 4.7, and 4.9.

With the four basic Lotka-Volterra models, three composed models were created, one for each type of composition. Models were composed in a white-box integration manner; as how to generate composed models from components is not the focus of this thesis, no further discussion will be presented here. For the same prey composition, the model with foxes and rabbits was composed with the model with wolves and rabbits, and in the composed model both wolves and foxes hunt rabbits. Meanwhile, the model with wolves and sheep was composed with the model with wolves and rabbits for the same predator composition, and in the composed model wolves hunt both sheep and rabbits. Similarly, in the food chain composition, the model with wolves and foxes was composed with the model with foxes and rabbits, resulting in a composed model where wolves hunt foxes and foxes hunt rabbits. The reaction rules

```
Food:f + Rabbit:x -> Food + 2 Rabbit @b*#x;
Rabbit:x + Rabbit:z -> Rabbit @k*#z*#x;
Wolf:y + Rabbit:x -> 2 Wolf @a*#y*#x;
Wolf:y -> @d*#y;

Fox:y + Rabbit:x -> 2 Fox @a*#y*#x;
Fox:y -> @d*#y;
```

(a) Reaction rules of the composed model from the same prey composition. [From [184]]

```
Food:f + Rabbit:x -> Food + 2 Rabbit @b*#x;
Rabbit:x + Rabbit:z -> Rabbit @k*#z*#x;
Wolf:y + Rabbit:x -> 2 Wolf @a*#y*#x;
Wolf:y -> @d*#y;

Food:f + Sheep:x -> Food + 2 Sheep @b*#x;
Sheep:x + Sheep:z -> Sheep @k*#z*#x;
Wolf:y + Sheep:x -> 2 Wolf @a*#y*#x;
```

(b) Reaction rules of the composed model from the same predator composition.

```
Food:f + Rabbit:x -> Food + 2 Rabbit @b*#x;
Rabbit:x + Rabbit:z -> Rabbit @k*#z*#x;
Fox:y + Rabbit:x -> 2 Fox @a*#y*#x;
Fox:f -> @d*#f;

Wolf:w + Fox:y -> 2 Wolf @a*#y*#w;
Wolf:w -> @d*#w;
```

(c) Reaction rules of the composed model from the food chain composition. [From [184]]

Figure 4.10.: Reaction rules of composed Lotka-Volterra models.

of the three composed models are shown as follows: the same prey composition in Figure 4.10(a), the same predator composition in Figure 4.10(b), and the food chain composition in Figure 4.10(c).

As discussed in [183], to compose ML-Rules models, the modeler needs to decide whether variables from different model components but with the same name, refer to the same variable and should be merged in the composed model, or refer to different ones and thus should be renamed. In addition to renaming species, the modeler might also change the configuration of parameters during composition. For example, in the food chain composition, the initial population size of the species that is now both prey and predator, may need adjustment in the composed model. All this information goes into the adaptation information, which is needed for generating experiments for the composed model based on experiment specifications of model components. In this case study, no adaptation on parameter names was needed and for the food chain composition, the initial population size of the species fox was configured to 30.

	Same Prey	Same Predator	Food Chain
Coexisting state	√ / √	⊘ / ⊘	√ / ⊘
Empty state	√ / √	√ / √	⊘ / √
Recovery comparison	√ / √	⊘ / √	√ / ⊘

Table 4.1.: Results overview for the case study of Lotka-Volterra models. For each composition type, the experiment specifications of the two model components are checked. Each cell contains the results of checking experiment specifications from both components that refer to the same property. The symbol √ represents that the property holds for the composed model, whereas the symbol ⊘ indicates that the property does not hold. [From [184]]

4.4.5. Reusing Simulation Experiments of Model Components

As a result of three types of composition, three composed models have been created, i.e., the same prey composition, the same predator composition, and the food chain composition. In each composition, two basic Lotka-Volterra models were reused and each model was annotated with three experiment specifications, corresponding to three properties, i.e., the "coexisting state", the "empty state", and the "recovery comparison". Thereby, for each composed model, altogether six experiment specifications from both model components were reused to generate experiments and check the corresponding property.

As all properties specified in experiment specifications involve probability estimation, the method described in Algorithm 2 (Section 4.4.2, pp. 89-90) was used to determine the number of necessary replications. The adaptation method based on generating parameter assignment ranges (see Section 4.3.2, pp. 84-85) has been used to generate new experiments, and for each generated parameter assignment range, 10 values uniformly sampled from the range were used for experiment execution. Thereby, all experiment specifications of model components were reused, adapted and executed for the composed model, and the experiment results were checked against specified properties.

An overview of the checking results is given in Table 4.1, where √ means that the property holds in the composed model, and ⊘ that it does not hold (at least not with the expected probability $p \geq 0.8$). For each type of composition, the properties of the two reused models are checked for the composed model.

Looking at the results, one can see that the same prey composition, where two predators feeding on the same prey, is unproblematic. This is as expected, as the prey controls the behavior of the predator(s) in the model. The situation is different for the same predator composition, where a predator feeds on two prey species, as the weaker prey is likely to become extinct, which can be reflected by the fact that in the composed model the property "coexisting state" does not hold and "empty state" holds. In the case of food chain composition, for each property, only experiment specifications of one model component can be checked successfully in the composed model, i.e., in each cell from the last column of Table 4.1 only one $\surd$ exists. This indicates that this composition type might not be valid. Actually, in the given example it would be possible that the wolves to also feed on rabbits as soon as rabbits are around, and not only on foxes (cf. Figure 4.10(c)).

Table 4.2 shows the results in more detail. Here, an interesting observation is that the property "Recovery comparison" for the same predator composition holds for 9 out of 10 tested assignments in the composed model, in which case the test result is false. It would be difficult to find the counterexample, if the properties were checked manually. This illustrates the potential of the presented approach in preventing users from coming to incorrect conclusions, by automatically generating simulation experiments for the composed model and evaluating experiment results against the properties.

Original statement	Composition Type	Adaptation Information	$\lvert A_+\rvert$	$\lvert A_-\rvert$	Counterexample
Coexisting State(Rabbit, Fox)	The Same Prey	–	10	0	–
Coexisting State(Rabbit, Wolf)	The Same Prey	–	10	0	–
Coexisting State(Rabbit, Wolf)	The Same Predator	–	3	7	a:0.015;b:0.7;d:0.5; nRabbit:100;nWolf:10;nSheep:100
Coexisting State(Sheep, Wolf)	The Same Predator	–	1	9	a:0.012;b:0.9;d:0.5; nRabbit:100;nWolf:10;nSheep:100
Coexisting State(Rabbit, Fox)	The Food Chain	nFox:30	10	0	–
Coexisting State(Fox, Wolf)	The Food Chain	nFox:30	5	5	a:0.014;b:0.7;d:0.9; nRabbit:100;nWolf:10;nFox:30
Recovery Comparison(Rabbit, Fox)	The Same Prey	–	10	0	–
Recovery Comparison(Rabbit, Wolf)	The Same Prey	–	10	0	–
Recovery Comparison(Rabbit, Wolf)	The Same Predator	–	9	1	a:0.024;b:0.9;d:0.5; nRabbit:10;nWolf:10;nSheep:10
Recovery Comparison(Sheep, Wolf)	The Same Predator	–	10	0	–
Recovery Comparison(Rabbit, Fox)	The Food Chain	–	10	0	–
Recovery Comparison(Fox, Wolf)	The Food Chain	–	10	0	–
Empty State(Rabbit, Fox)	The Same Prey	–	10	0	–
Empty State(Rabbit, Wolf)	The Same Prey	–	10	0	–
Empty State(Rabbit, Wolf)	The Same Predator	–	10	0	–
Empty State(Sheep, Wolf)	The Same Predator	–	10	0	–
Empty State(Rabbit, Fox)	The Food Chain	nFox:30	0	10	a:0.055;b:0.3;d:0.8; nRabbit:100;nWolf:30;nFox:30
Empty State(Fox, Wolf)	The Food Chain	nFox:30	10	0	–

Table 4.2.: Detailed results for the case study of Lotka-Volterra models. The column "Original statement" shows the properties and the related species. All properties are checked regarding their probability ($p \geq 0.8$), which is omitted for simplicity. $|A_+|$ and $|A_-|$ are the number of assignments for which the property holds and does not hold, respectively. The sampling number for each parameter assignment range is set to 10, thus $|A_+| + |A_-| = 10$. Only when experiment results for all 10 sampled assignments are checked successfully, the test result is true. For each experiment specification where the specified property is not checked successfully for the composed model, a counterexample of parameter assignment is given (see rows with gray background). [Adapted From [184]]

4.5. Case Study II: Modeling the Wnt/β-catenin Pathway

While the previous case study based on a small model of Lotka-Volterra mainly serves as a demonstration of how the proposed approach is used to support model composition, in this case study, the proposed approach was applied to build a realistic model the Wnt/β-catenin pathway by successively composing three individual models. By following the life cycle of model composition (Figure 4.1), this case study also aims to illustrate where the presented approach is located in this process and how the approach can be combined with other existing work on model composition.

The simulation study described in this case study had been performed by Haack [87], with two models of Wnt/β-catenin pathway already developed (see [89]). To demonstrate the applicability of the proposed approach, this case study replayed the simulation study and reconstructed the model development process explicitly in a composition way.

4.5.1. Background

Wnt/β-catenin signaling is involved in central cellular processes, such as differentiation, proliferation, and migration of cells. Its deregulated form leads to developmental disorders and various diseases, including several forms of cancer and Alzheimer's disease [39]. A better understanding of this pathway and its regulation of human neural progenitor cells (hNPCs) differentiating into neurons, astrocytes and oligodendrocyte cells allows more effective replacement therapies of neuro-degenerative diseases, such as Parkinson's or Huntington's disease. Consequently, the analysis of the Wnt/β-catenin signaling pathway has been the subject of a series of simulation studies and quantitative models [137]. Thus, modeling efforts often start with an existing model being revised, extended, or composed with other models, e.g., [89].

The key component of Wnt/β-catenin signaling is a protein termed β-catenin. In the inactive state of Wnt/β-catenin signaling, β-catenin is constantly produced, but immediately targeted for degradation by a large protein complex termed the destruction complex. Upon Wnt stimulation, a reaction cascade is triggered, which leads to the inhibition of major components of the destruction complex, such as Axin. As a result, β-catenin accumulates inside the cytosole and subsequently shuttles into the nucleus. Once shuttled into the nucleus, β-catenin associates with the Lef/Tcf transcription factors and triggers a pathway-specific gene response relevant for the

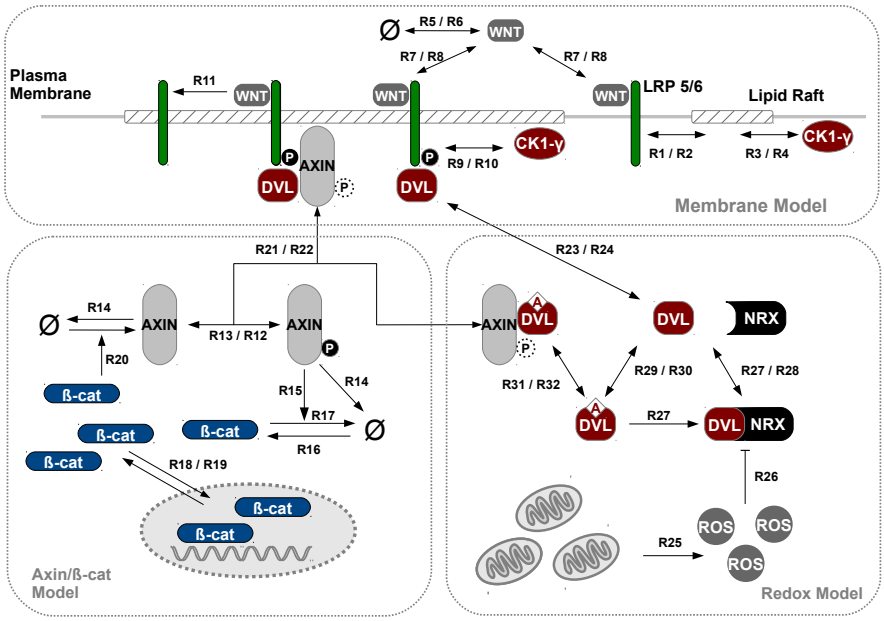


Figure 4.11.: The Wnt pathway model that contains three individual models: the membrane model, the Axin/β-cat model and the ROS model [89].

regulation of various physiological and developmental processes [39]. The relocation of β-catenin into the nucleus is therefore the main indicator for the pathway activation.

To study the regulation of β-catenin levels in response to various stimuli, a computational stochastic model of Wnt/β-catenin pathway has been developed by Haack et al. [89]. In this case study, the development of this model was replayed, and the process of developing models by composition and the role of reusing simulation experiments in this process were made explicit. Three individual models have been reused and successively composed: a membrane model (M1), an Axin/β-catenin model (M2) and a Reactive Oxygen Species (ROS) model (M3), as shown in Figure 4.11. Firstly, the membrane model is composed with the Axin/β-catenin model, which results into the first composed model (M12). Afterwards, the resulting composed model is further composed with the ROS model (M3), leading to the second composed model (M123), which consists of all three models.

4.5.2. Ingredients

Modeling and Simulation Method

A previous study of Wnt/β-catenin signaling pathway of human neural progenitor cells has already shown that stochastic effects should not be ignored [157]. Moreover, hierarchical structures exist in the system to be modeled, e.g., the cell includes nucleus and membrane, which are compartments containing further structures. Thus, a modeling language, which allows the representation of a formal, stochastic semantics

and the description of inter- and intra-compartmental dynamics, is needed. Therefore, ML-Rules was chosen to define models in this case study (see description in Section 4.4.2, pp. 87-88).

A simulation system for ML-Rules models, which has been integrated into SESSL via the binding `sessl.mlrules`, was employed to execute experiments, and the algorithm presented in [261] was used, named as `SimpleSimulator` in SESSL.

Property Specification and Checking Algorithms

Despite the fact that LTL has been successfully used in many applications, the temporal operators in the language do not allow for quantitative statements about time. In this case study, explicitly relating a certain observation with time is required for expressing the properties of interest. Thus, the Metric Interval Temporal Logic ($\text{MITL}_{[a,b]}$) [149] was employed, which enriches traditional modal operators of temporal logics with intervals and defines that the operator is only evaluated in this interval. For instance, $\phi \, \boldsymbol{U}_{[\mathbf{3,7}]} \, \psi$ states that there exists a time point t', which is between 3 to 7 time units from the current time point t, so that the property ψ holds from t' on and ϕ should hold in $[t, t']$.

The validation of models in this case study depended on checking simulation results against properties that are derived from observations in wet-lab experiments with human neural progenitor cells [89, 156]. The available data refer mostly to aggregate observations made on heterogeneous populations of thousands up to a million of cells, and those aggregate observations form the basis for further discussions and explorations. Given that each developed model in this case study refers to a single cell and no interaction between different cells is involved, the property checking also was based on averages of multiple stochastic simulation runs executed with the same model. This method differs from statistical model checking in its handling of variance among simulations: averaging over multiple runs instead of hypothesis testing on individual runs. The later was shown to be less suitable in this case study, as most of the properties that are of interest could not be verified by statistical model checking due to the high variance among different simulation runs. The variance can be reduced by the averaging of individual runs.

Since the wet-lab data used are based on average of observations from thousands up to a million of cells, to efficiently perform experiments while ensuring the accuracy of result analysis, a mechanism to determine the minimal number of required simulation replications is needed. Therefore, a method based on constraining the confidence interval was used, i.e., at each relevant observation point the confidence interval is calculated and the interval width is compared to a given threshold. Experiments

are executed in batches; after each batch, confidence intervals are calculated based on all executed replications. The same simulation set-up is replicated until at every observation point the calculated confidence interval has a width less than the defined threshold; otherwise, another batch of experiments will be scheduled. Once a sufficient number of runs are executed, a point-wise average of all simulations is calculated to represent the trajectory of the average cell, and this trajectory is used for checking properties. The confidence level, interval width, and batch number can be configured by the user. For experiments in this case study, the 95% confidence level was used and the width threshold was set to 10, i.e., the 95% confidence interval is constructed with the average of observed species number plus/minus at most 5, and one batch of experiments was configured to contain 5 replications.

Averaging across different simulation runs considers the variance between those runs. However, stochastic fluctuations over time exist in each simulation run and these fluctuations of single runs are still evident in the average trajectory. Specifications in $\text{MITL}_{[a,b]}$ and the corresponding checking algorithms are not able to tolerate such fluctuations. For example, an algorithm checking a trajectory for an increase might reject it because of some short, random episodes of decreasing. To avoid such errors, a preprocessing of trajectories was employed based on the LOESS smoothing method, which is a generalization of LOWESS (Locally Weighted Scatterplot Smoothing) originally proposed by Cleveland [38]. An implementation of the LOESS method from the Apache Commons Math library [40] was adopted and integrated into SESSL. The parameters bandwidth and iteration number of the LOESS algorithm can be configured in the SESSL specification, and the accuracy is set to 10^{-12} in the implementation. In this case study, the LOESS method was configured with a bandwidth of 0.1 and no additional iterations, constituting a minimal smoothing.

The algorithm proposed in [149] was exploited for checking $\text{MITL}_{[a,b]}$ formulas. To obtain the valid intervals for a given formula, the valid intervals for its subformulas are recursively determined (an example is shown in Figure 4.12). The logical operators negation, conjunction and disjunction are mapped to inverting, intersecting and joining the sets of valid intervals. The valid intervals for atomic propositions are determined directly from the trajectory data. If one of these intervals contains the time point 0, i.e., the beginning of the trajectory, the formula holds for the observed simulation run.

Special attention has to be given to the semantics of these intervals. As defined by Maler and Nickovic, the intervals are interpreted as left-closed right-open, such that they contain their start point, but not their endpoint [149]. Although this would mathematically be denoted as $[t_i,t_{i+1})$, here $[t_i,t_{i+1}]$ is used following the original

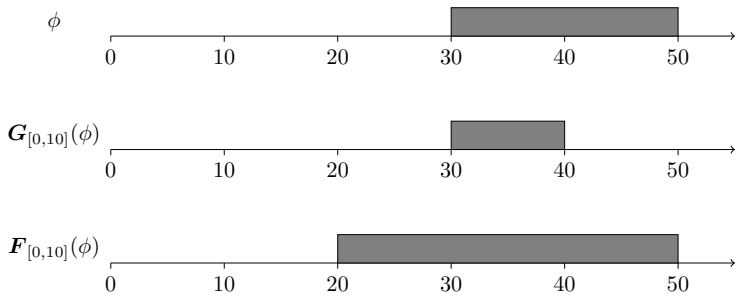


Figure 4.12.: Example for obtaining valid intervals for formulas with temporal operators. The valid intervals for the formulas $\boldsymbol{G}_{[0,10]}(\phi)$ (ϕ holds during the next ten time units) and $\boldsymbol{F}_{[0,10]}(\phi)$ (ϕ holds somewhere in the next ten time units) should be determined. The formula ϕ holds in the interval [30,50]. Then the formula $\boldsymbol{G}_{[0,10]}(\phi)$ $[30, 40]$, and $\boldsymbol{F}_{[0,10]}(\phi)$ holds in $[20, 50]$. Picture after Maler and Nickovic [149]. [From [186]]

authors. The set of observed time points in the trajectory makes up possible interval borders. Assuming that the model state does not change between observations regarding the property to check, a property holds in an atomic interval $[t_i,t_{i+1}]$ if it holds at the observed time point t_i. Consequently, the observed trajectory should contain enough data points to justify this assumption. As "enough" is highly model-dependent, it is crucial to interweave the configuration of model observation and MITL$_{[a,b]}$ property checking in the experiment specification, which is provided by SESSL.

SESSL was extended to support those new features, i.e., determining required simulation replications based on thresholds for point-wise confidence interval width, computation of the average trajectory, data preprocessing with LOESS trajectory smoothing, and model behavioral property specification and checking based on MITL$_{[a,b]}$. For experiment adaptation and generation, the first method discussed in Section 4.3.2 is used, i.e., a set of default configurations for all model parameters is provided by the user in the composed model file and for those parameters that should be limited to a specific configuration, their configuration is defined as part of adaptation information.

4.5.3. Developing the Wnt/β-catenin Model

Problem Formulation, Requirements Engineering, and Conceptual Modeling

As the starting point of a simulation study, this stage leads to the development of a conceptual model. According to the definition from Robinson [200], a conceptual

model typically refers to *objectives, inputs, outputs, content, assumptions and simplifications*. Based on this definition, the conceptual model in this case study was presented as follows.

An involvement of lipid rafts in Wnt/β-catenin signaling has been shown in several studies [211]. However, the spatio-temporal dynamics and the exact impact of lipid rafts on Wnt/β-catenin signaling remain unclear. Therefore, the objective of the simulation study, as presented by Haack et al. [89], was to explore the role lipid rafts play in the Wnt/β-catenin signal transduction, with the presence/removal of lipid rafts as the main input and the concentration of nuclear β-catenin as the output. This motivated the development of a Wnt/β-catenin model with membrane related dynamics considered as well.

The Wnt/β-catenin signaling starts with Wnt binding to the receptor LRP6 to form the Wnt receptor complex. The phosphorylation of the complex is restrained to be inside lipid rafts, and the phosphorylated complex recruits the key component of the destruction complex Axin, and therefore causes an accumulation of β-catenin inside the nucleus.

One simplification is that only LRP6 and CK1γ are considered for the receptor complex. As a matter of fact, the Wnt receptor complex also comprises a sub-variant of the Frizzled receptor as well as additional membrane-bound adapter proteins. This simplification is reasonable since for Wnt/β-catenin signaling, all crucial events mainly depend on the dynamics of LRP6 and CK1γ. In addition, only Axin is considered as the destruction complex, and the remaining proteins such as GSK3β are disregarded. Mathematical analyses have shown that such a reduced model is capable of reproducing the essential dynamics of Wnt-induced β-catenin signaling [164]. As the wet-lab data refer to the first 12 hours (720 minutes) of differentiation, simulation experiments of models in this study have also been constrained to this time period, with 1 minute in the wet-lab experiments corresponding to 1 simulation time unit.

Based on this conceptual model, the aim was to compose a membrane model that comprises central key players of membrane related dynamics such as the receptor complex and its interaction with lipid rafts, and a core model of the Wnt/β-catenin that is able to predict the β-catenin accumulation within the nucleus depending on the activity of the destruction complex.

Model Selection — the Membrane Model (M1)

In the ideal situation, a suitable model can be found and retrieved from a model repository. Unfortunately, it is often the case that experimental studies provide new

insights or evidence into biochemical or cell-biological processes that are not yet covered by existing models. Therefore, a new model has to be developed, as was the case in the simulation study in [89]. A membrane model (M1) was developed to represent the diffusion-driven shuttling of LRP6 and CK1γ between raft and non-raft membrane regions, where lipid rafts are modeled as individual compartments within the membrane, similar to the nucleus being a single compartment within the cell. Thereby, the fraction of raft-associated LRP6 (LR[Lrp6]) and CK1γ (LR[CK1γ]) molecules are the main behavioral properties for the membrane model.

Throughout this case study, the values for model parameters and initial concentrations were partly fitted, partly taken from literature, and partly directly measured in wet-lab experiments (see Table 1 in [89] for more detail). The model parameters were configured as $\texttt{nLR} = 5, \texttt{nLrp6} = 4000, \texttt{nCK1y} = 5000$, where `nLR, nLrp6, nCK1y` denote the initial number of lipid raft, LRP6, and CK1γ, respectively. Using the simulation method presented in Section 4.5.2, simulation experiments were performed with this model, and during the simulation the number of species LR[Lrp6] and LR[CK1γ] have been observed and recorded. To alleviate stochastic fluctuations, the LOESS method was used to smooth generated simulation trajectories. The simulation trajectories are shown in Figure 4.13. The LOESS method may have an impact on the initial values, peaking values, or valley values in the trajectories, such as increasing the values (as shown in Figure 4.13). However, this increasing of values does not affect the model behavioral properties that are of interest. This also applies to the experiments with other models in this case study, e.g., Figure 4.15, where in the beginning part of the trajectory the value of the first peak and valley slightly deviate from original ones.

Based on the findings in wet-lab studies [211] and the simulation results, two properties were derived: after a warm up phase, the fraction of LRP6 and CK1γ being associated with lipid rafts are in equilibrium, with around 1/4 of the overall Lrp6 population and around 3/4 of CK1γ population residing within lipid rafts. Given the model $M1$ with its parameters $\tilde{p}$ and the simulation set-up $ses1$, the above properties can be specified in $\text{MITL}_{[a,b]}$ as follows:

$$M1_{\tilde{p}} \models_{ses1} G(60,\ 720,\ LRLrp6 >= nLrp6 * 0.25\ \wedge LRLrp6 <= nLrp6 * 0.3) \tag{4.1}$$

$$M1_{\tilde{p}} \models_{ses1} G(60,\ 720,\ LRCK1\gamma >= nCK1\gamma * 0.7 \wedge LRCK1\gamma <= nCK1\gamma * 0.75) \tag{4.2}$$

The experiments were specified in SESSL, including the desired behavioral properties, as presented in Figure 4.14. Line 10 denotes the specification of replication conditions that for both species `LR[Lrp6]` and `LR[CK1`γ`]`, the average value observed at every 120 time units from time 0 to 720 should have 95% confidence interval with

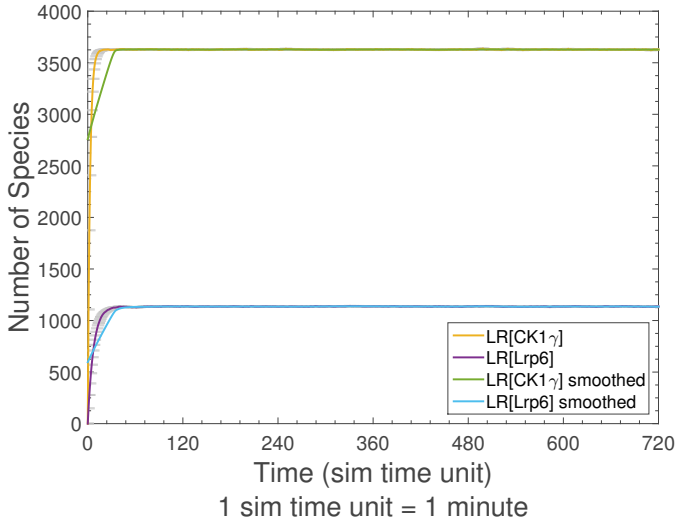


Figure 4.13.: Simulation results of the Membrane Model (M1), with the average trajectories with 95% confidence interval (gray error bars) and the smoothed trajectories. Please note, as a result of the smoothing, the initial values of trajectories slightly deviate from the original trajectories. As explained in the text, this deviation does not affect the considered model behavioral properties, hence does not bias test results.

width less than 10, and 5 replications constitutes one experiment batch. Line 11 configures the data preprocessing method, i.e., the LOESS smoothing method. The corresponding property states that for the average of all simulated trajectories, from time 60 to time 720, the amount of `LRLrp6` is always between 25% and 30% of the initial amount of Lrp6 (i.e., 4000), and the amount of `LRCK1y` is always between 70% and 75% of the initial amount of CK1γ (i.e., 5000) (lines 12-15).

The experiment specification was successfully tested with the membrane model using the developed tool TAECS without adaptation, which corresponds to the working case B in Figure 3.9, and thereby annotated with the membrane model M1.

Model Selection — the Axin/β-Catenin Model (M2)

Several models can be found in public repositories (such as BioModels Database) that describe the core β-catenin dynamics [137]. However, only one model is of stochastic nature and had been validated for hNPCs [157], which therefore was selected for composition.

This intracellular model primarily describes the dynamics of β-catenin, including its synthesis and its interaction with the destruction complex and the resulting degradation process, as well as its shuttling between nucleus and cytosol. These dynamics are triggered by an initial amount of Wnt. The model resembles a reduced

```
import sessl._
import sessl.mlrules._
val exp = new Experiment with Observation with Hypothesis {
 model = "./MembraneModel.mlrj"
 set("nLR" <~ 5, "nLrp6" <~ 4000, "nCK1y" <~ 5000, "nWnt" <~ 220)
 simulator = SimpleSimulator()
 stopTime = 720.0
 observe("LRLrp6" ~ "Cell/Membrane/LR/Lrp6", "LRCK1y" ~ "Cell/Membrane/LR/CK1y")
 observeAt(range(0.0, 1.0, 720.0))
 replicationCondition = PointWiseConfidenceInterval("LRLrp6", 5, 10.0, 0.95, range(0, 120,
     720)) and PointWiseConfidenceInterval("LRCK1y", 5, 10.0, 0.95, range(0, 120, 720))
 dataProcessor = LoessInterpolation(0.1, 0)
 assume { Average (
  G(60.0, 720.0, (variable("LRLrp6") >= 4000*0.25) and (variable("LRLrp6") <= 4000*0.3))
   and G(60.0, 720.0, (variable("LRCK1y")>=5000*0.7) and variable("LRCK1y")<=5000*0.75))
  )}
}
```

Figure 4.14.: Experiment specification of the Membrane model (M1) in SESSL.

version of the Lee model [127] and contains only two proteins (β-catenin and Axin). Axin is further characterized by a phosphorylation site, which is important for the degradation of β-catenin. While phosphorylated Axin (AxinP) promotes β-catenin degradation, its unphosphorylated state (Axin) has no impact on β-catenin. The model has been validated based on in-vitro data of hNPC [155]. To reproduce the biphasic behavior observed in the wet-lab, a delayed autocrine mechanism proved to be essential and had been introduced into the model [157].

The model was originally represented in SBML; as ML-Rules was chosen for model description (see Section 4.5.2), the model has been converted into ML-Rules. Since no experiments had been associated with the model, simulation experiments were reconstructed based on the publication [157]. The simulation results are shown in Figure 4.15. From the results, one can observe that within the first 2 hours of differentiation a peak occurs, after which β-catenin increases slightly again and stabilizes around a 1.4-1.7 fold increase. This biphasic behavior of the nuclear β-catenin concentration is the main desired property for this model, which can be expressed based on $\text{MITL}_{[a,b]}$ as follows:

$$M2_{\tilde{p}} \models_{ses2} (d(\beta cat)/dt >= 0)\ U(40,\ 100,\ F(0,\ 10,\ d(\beta cat)/dt <= 0)) \tag{4.3}$$

$$M2_{\tilde{p}} \models_{ses2} G(400,\ 720,\ \beta cat >= 1.4 * n\beta cat\ \wedge\ \beta cat <= 1.7 * n\beta cat) \tag{4.4}$$

The experiments have been specified in SESSL and successfully tested, as depicted in Figure 4.16, with Figure 4.16(a) referring to property in Equation 4.3 and Figure 4.16(b) referring to property in Equation 4.4.

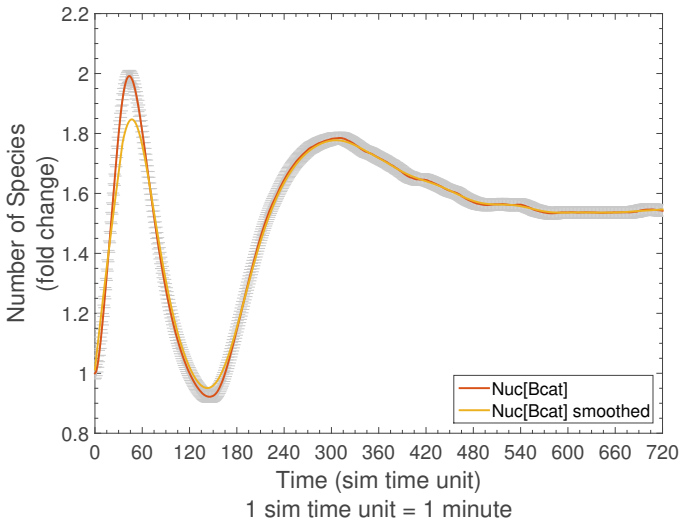


Figure 4.15.: Simulation results of the Axin/β-Catenin Model (M2), with the average trajectories with 95% confidence interval (gray error bars) and the smoothed trajectories.

```
val exp = new Experiment with Observation with Hypothesis {
 model = "./AxinBcatModel.mlrj"
 set("nBcatNuc" <~ 5282)
 simulator = SimpleSimulator()
 stopTime = 720.0
 observe("NucBcat" ~ "Cell/Nuc/Bcat")
 observeAt(range(0.0, 1.0, 720.0))
 replicationCondition=PointWiseConfidenceInterval("NucBcat", 5, 200.0, 0.95, range(0, 120,
     720))
 dataProcessor = LoessInterpolation(0.1, 0)
 assume { Average (
  d("NucBcat") >= 0 U(40.0, 100.0, F(0.0, 10.0, d("NucBcat") <= 0))
  )}
}
```

(a) Experiment specification regarding regarding the fold change of nuclear β-catenin concentration peaking within first 100 minutes.

```
assume { Average (
 G(400.0, 720.0, (variable("NucBcat") >= 1.4 * 5282) and (variable("NucBcat") <= 1.7 * 5282))
 )}
```

(b) Property specification regarding the fold change of nuclear β-catenin concentration staying in a certain range in the later hours. The experiment set-up is the same as that in (a).

Figure 4.16.: Experiment specifications of the Axin/β-Catenin model (M2) in SESSL.

Composing M1 and M2 — the Composed Model (M12)

A white-box integration approach was used to compose the membrane model M1 and the Axin/β-Catenin model M2, where the variables, structures, and rules of the reused models are merged (see discussion in Section 4.2, pp. 77-78), leading to the composed model M12. So far, the generation of a composed model from two ML-Rules models is realized by the user, as it is rather difficult (if not impossible) to automatically compose two models based on fusion. In the composed model, due to membrane related processes, the species Wnt no longer directly interacts with Axin as in the model M2, but is mediated: Wnt-bound and phosphorylated LRP6 (Lrp6PP) recruit and bind Axin at the membrane. Thus, LRP6 interferes with the function of Axin in the degradation complex, and hence disrupts the degradation of β-catenin. Besides, the phosphorylation of Wnt-bound Lrp6 is confined to lipid rafts [176]. Only phosphorylated Lrp6 recruits Axin from the cytosol, and due to its size, the resulting signalosome is unlikely to leave the lipid raft, thus having an impact on the diffusion patterns of Lrp6.

Therefore, the rule relating Wnt and Axin had to be removed, and additional rules were introduced to describe the interaction between Wnt and Lrp6, the phosphorylation of Lrp6 , and the interaction between Lrp6 and Axin, respectively. Parameters that are related to Wnt (as Wnt is no longer directly interacting with Axin) or to newly added rules, were (re-)calibrated based on recent wet-lab data or based on literature [89], and the rest were inherited from the two composed models.

Syntactic Verification

The previously described two models (M1 and M2) are independent, i.e., they share neither rules nor do they have common species; therefore, no merging conflicts exist in model variables. The syntactic checking of the composed model relied on the user, and an ML-Rules model editor had been developed to provide some assistance through syntax highlighting, which ensures the composed model is syntactically correct, and types and variables adhere to the constraints provided.

Semantic Validation

To check validity of the composition with the proposed approach, and the experiment specifications of M1 and M2 (as shown in Figure 4.14 and 4.16) were reused to conduct experiments with the composed model M12. No renaming of model parameters was performed during the composition and the configuration of the newly introduced model parameters had been provided within the model file by the user. Thus, in the

reused experiment specifications only the model location needed to be updated, from model component M1 or M2 to the composed model M12.

As already stated, in the presence of Wnt and the forming of the signalosome, an equilibrium of Lrp6 within lipid rafts would not be expected; instead, the fraction of raft-associated LRP6 would increase due to the size and reduced diffusion speed of the signalosome [88]. Thus, the property regarding the fraction of LRP6 within lipid rafts, i.e., Equation 4.1, would be expected to not hold for the composed model. As CK1γ is not influenced in the composition, its dynamics remains the same, i.e., Equation 4.2. Moreover, the properties specified in experiment specifications of the model M2 refer to β-catenin (see Equation 4.3 and 4.4), which were derived based on validation of M2 against wet-lab data; therefore, they should also hold for M12.

With the developed tool TAECS, the experiment specifications of M1 and M2 were adapted and executed on the composed model, and the behavioral properties were checked accordingly. The simulation trajectories are shown in Figure 4.17(a) and 4.17(b). The simulation results show that all properties of the reused models, except for the one referring to the stable fraction of raft-associated Lrp6 concentration, hold for the composed model. This was in agreement with the expectation, indicating that the composed model M12 indeed works as intended with respect to the key behavioral properties of its two reused models.

Therefore, the composed model M12 possesses the three behavioral properties from its two model components under corresponding experiment set-ups, i.e., the properties regarding species CK1γ (Equation 4.2, from M1) and nuclear β-catenin (Equation 4.3 and 4.4, from M2), which can be expressed based on $\text{MITL}_{[a,b]}$ as Equation 4.6, 4.7 and 4.8. As for the property regarding the species Lrp6, the composed model M12 behaves differently from the model M2, i.e., the amount of Lrp6 within lipid rafts accumulates instead of reaching a stable fraction, as illustrated by the simulation results in Figure 4.17(a) and Figure 4.13. Therefore, this property was updated, as described in Equation 4.5, while the experiment set-up stays the same as that in the original model M1.

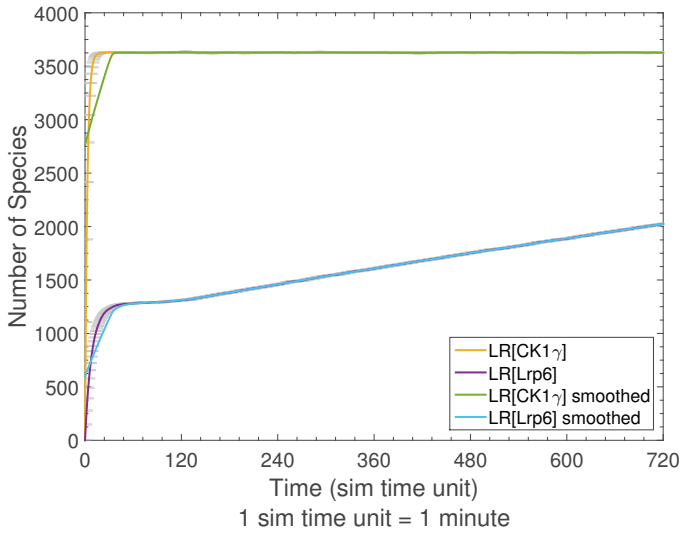
$$M12_{\tilde{p}} \models_{ses1} G(0, 720, d(LRLrp6)/dt >= 0) \tag{4.5}$$

$$M12_{\tilde{p}} \models_{ses1} G(60,\ 720,\ LRCK1\gamma >= nCK1\gamma * 0.7\ \wedge\ LRCK1\gamma <= nCK1\gamma * 0.75) \tag{4.6}$$

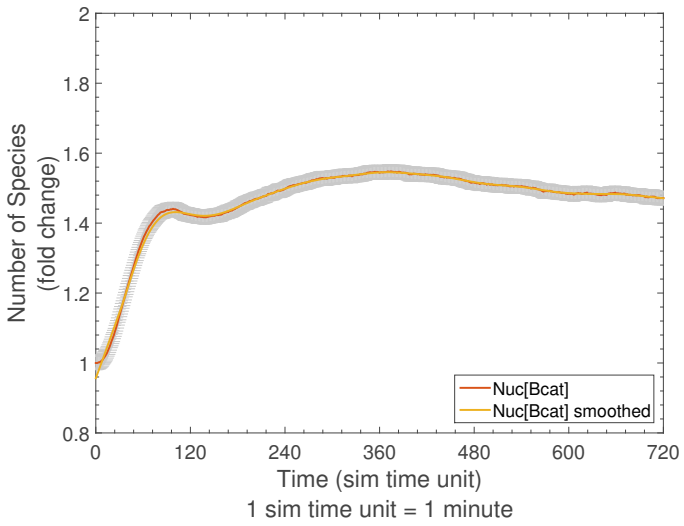
$$M12_{\tilde{p}} \models_{ses2} (d(\beta cat)/dt >= 0)\ U(40,\ 100,\ F(0,\ 10,\ d(\beta cat)/dt <= 0)) \tag{4.7}$$

$$M12_{\tilde{p}} \models_{ses2} G(400,\ 720,\ \beta cat >= 1.4 * n\beta\ \wedge\ \beta cat >= 1.7 * n\beta cat) \tag{4.8}$$

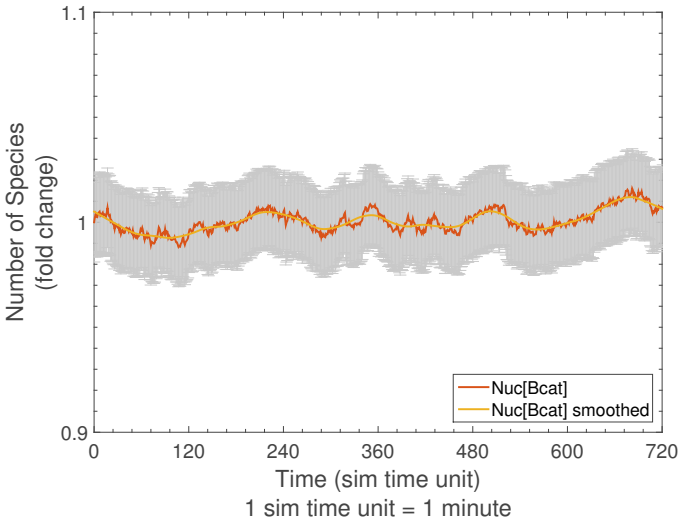
Additional simulation experiments were executed to further validate the composed model M12, e.g., to reproduce an independent study in [91], different amounts of



(a) The simulation results regarding the species Lrp6 and CK1γ inside lipid rafts.



(b) The simulation results regarding nuclear β-catenin with the default model configuration.



(c) The simulation results regarding nuclear β-catenin when lipid rafts are removed, i.e., `nLR` $= 0$.

Figure 4.17.: Simulation results of the first composed model (M12), with the average trajectories with 95% confidence interval (gray error bars) and the smoothed trajectories.

initial Wnt were applied to the model and the resulting nuclear β-catenin levels were compared. Subsequently, the question that motivated this simulation study was examined, namely analyzing the impact of lipid rafts on β catenin based on the composed model M12. Within the wet-lab experiments, the lipid rafts of the hNPCs were disrupted by applying methyl-beta-cyclodextrin. Whereas the Wnt-β-catenin signalling pathway appeared inactive in the later hours as expected (i.e., no β catenin fold increase), a peak of β-catenin within the first three to four hours were observed.

Two properties were derived based on this observation, which can be expressed with $\text{MITL}_{[a,b]}$ as in Equation 4.10 and 4.9, while the experiment set-up $ses12$ is similar to $ses2$ except for nLR = 0. To reproduce the findings of the wet-lab experiments, the lipid rafts were accordingly removed from the model M12, by setting nLR = 0. Simulation experiments were executed and results are presented in Figure 4.17(c), showing that the property in Equation 4.10 cannot be satisfied by the composed model M12, only the one in Equation 4.9.

$$M12_{\tilde{p}} \models_{ses12(nLR=0)} G(400,\ 720,\ \beta cat >= 0.9 * n\beta cat\ \wedge\ \beta cat <= 1.1 * n\beta cat) \tag{4.9}$$

$$M12_{\tilde{p}} \models_{ses12(nLR=0)} (d(\beta cat)/dt >= 0\, U(40,\ 200,\ F(0,\ 10,\ d(\beta cat)/dt <= 0)) \tag{4.10}$$

Taking a closer look at the composed model M12, one can find the influence of the Wnt on β-catenin entirely relies on lipid rafts; when completely removing lipid rafts from the model, the raft-dependent LRP6 phosphorylation by CK1γ in response to a Wnt stimulus is prevented, and no Axin can be recruited, which results in no accumulation of β-catenin inside the nucleus. However, this disagrees with the observation in wet-lab experiments, indicating a missing mechanism in the composed model M12 that leads to the immediate increase of β-catenin and its peak in raft deficient cells within the first 3 hours.

Model Documentation and Storage

Despite the fact that the composed model M12 is not able to reproduce all observations from wet-lab data, it exhibits valid behavior under certain conditions. Therefore, to facilitate further reuse, M12 was annotated with those experiments that have been successfully checked, as shown in Figure 4.18, including the experiment from model M1 with property regarding the species CK1γ, the experiment from model M1 with an updated property regarding the species Lrp6, the two experiments from model M2 with properties regarding the nuclear β-catenin, and the experiment with property regarding the nuclear β-catenin in the later hours under the condition that lipid rafts are removed.

```
assume { Average (
 G(60.0, 720.0, variable("LRCK1y") >= 5000 * 0.7 and (variable("LRCK1y") <= 5000 * 0.75))
  and G(0.0, 720.0, d("LRLrp6") >= 0)
 )}
```

(a) Property specification regarding the fraction of LR[Lrp6] and LR[CK1γ]. The first order derivative is used to describe that a variable increases or decreases, denoted as d("variableName") in SESSL.

```
val exp = new Experiment with Observation with Hypothesis {
 model = "./FirstComposedModel.mlrj"
 set("nBcatNuc" <~ 5282, "nLR" <~ 0)
 simulator = SimpleSimulator()
 stopTime = 720.0
 observe("NucBcat" ~ "Cell/Nuc/Bcat")
 observeAt(range(0.0, 1.0, 720.0))
 replicationCondition=PointWiseConfidenceInterval("NucBcat", 5, 200.0, 0.95, range(0, 120,
     720))
 dataProcessor = LoessInterpolation(0.1, 0)
 assume { Average (
  G(400.0, 720.0, (variable("NucBcat") >= 0.9 * 5282) and (variable("NucBcat") <= 1.1 * 5282))
  )}
}
```

(b) Experiment specification regarding the fold change of nuclear β-catenin concentration when lipid rafts are removed from the model by configuring `nLR = 0`.

Figure 4.18.: Experiment specifications of the composed model (M12) in SESSL. (a) depicts the updated property specification regarding LR[Lrp6] and LR[CK1γ] while the experiment set-up is similar to Figure 4.14 except for the model location, and (b) describes the experiment specification regarding nuclear β-catenin under the condition that lipid rafts are removed. In addition, the first composed model is annotated with another two experiment specifications, i.e., the adapted experiment specifications from M2 as shown in Figure 4.16 by updating model location.

Further Composition — Integrating ROS/Dvl Model (M3)

In the presence of lipid raft the composed model M12 appears to be valid; in the absence of lipid rafts only its response referring to the last 8 hours of the experiments is as expected, at which time autocrine mechanisms are already effective. Thus, the model underlines the role of lipid rafts particularly in combination with autocrine mechanisms for the later hours of the early hNPCs' differentiation. Therefore, a new round of modeling within the life cycle of model composition was started to explore

possibilities that could explain the first peak. In the following, this study will be briefly presented, with a focus on the reuse of simulation experiments for semantic validation of the composed model.

Hence, a model was needed, which can be composed with the model M12 so that the resulting model can still exhibit the key properties of model M12 with corresponding experiments and additionally produce the expected behavior that under the condition lipid rafts are removed, an increase of nuclear β-catenin can be observed within the first hours.

A related study was identified, in which during the initiation phase of hNPCs' differentiation an early spontaneous production of reactive oxygen species (ROS) had been observed, and the ROS promotes a Dvl-mediated downstream activation of canonical Wnt signaling [197]. Therefore, the next model component was aimed at the ROS/DVL model (M3), with which the first composed model M12 would be further composed. Here the previously described model M12 of Wnt/β-catenin signaling pathway is complemented by an intracellular signaling mechanism that is completely independent of extracellular Wnt molecules. β-catenin signaling is initiated by a spontaneous release of Reactive Oxygen Species (ROS) from mitochondria, which in turn activates Dvl by releasing the redox-sensitive binding of NRX and DVL. Subsequently, cytoplasmic DVL binds to Axin and thereby leads to the activation of downstream β-catenin signal pathway, yielding a transient accumulation of β-catenin [89, 197].

Only few quantitative experimental data were available, and the model was based upon and calibrated against the findings of two publications [75, 197] and additional experimental data [89]. Simulation experiments were performed with the ROS model (M3) and the dynamics of DVL concentration were observed, and the simulation results are presented in Figure 4.19. According to the observation of the spontaneous increase in cytosolic DVL concentration in the wet-lab [197], the main behavioral property is that the DVL concentration reaches a value between 10000 and 12000 very fast directly at the beginning (assumed within the first 5 minutes). The property can be expressed based on $\text{MITL}_{[a,b]}$ as:

$$M3_{\tilde{p}} \models_{ses3} F(0.0,\ 5.0,\ Dvl >= 10000\ \wedge\ Dvl <= 12000) \tag{4.11}$$

As the fast peak of DVL concentration is the main property, an additional smoothing is not needed. The experiment was specified in SESSL, as shown in Figure 4.20.

By binding Dvl (from model M3) to both LRP6 and Axin (from model M12), the composed model M123 was created, with four rules introduced, and the kinetics of these new rules were defined based on wet-lab experiments and literature.

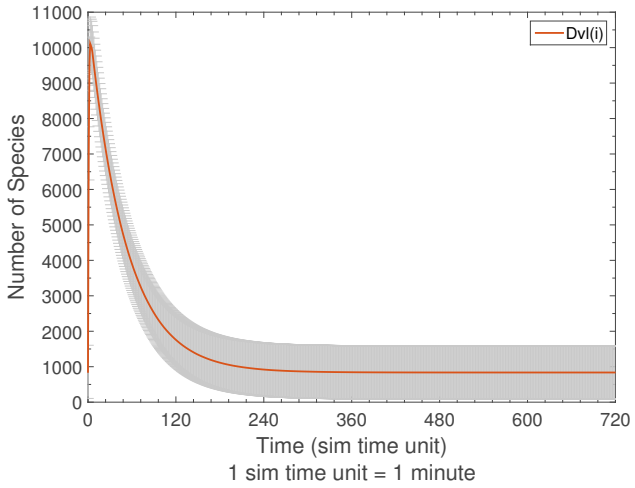


Figure 4.19.: Simulation results of the ROS model (M3) with the average trajectory with 95% confidence interval (gray error bars).

```
val exp = new Experiment with Observation and Hypothesis {
 model = "./RosModel.mlrj"
 simulator = SimpleSimulator()
 stopCondition = 720.0
 observe("Dvl" ~ "Cell/Dvl(i)")
 observeAt(range(0.0, 1.0, 720.0))
 replicationCondition = PointWiseConfidenceInterval("Dvl", 5, 1500.0, 0.95, range(0, 1, 5))
 assume { Average (
  F(0.0, 5.0, variable("Dvl") >= 10000 and (variable("Dvl") <= 12000))
  )}
}
```

Figure 4.20.: Experiment specification of the ROS model (M3) in SESSL.

Accordingly, the composed model M123 was evaluated by reusing the experiment specifications of the ROS/Dvl model component (M3) and the first composed model M12, assuming that none of the original dynamics are disrupted.

In the models M12 and M2, it was assumed that the peak of nuclear β-catenin within first hours is Wnt-dependent, i.e., the property described in Equation 4.7 and 4.3, corresponding to the experiment depicted in Figure 4.16(a). But this assumption appears to not hold according to the contradiction between simulation results and wet-lab observation when lipid rafts are removed. Therefore, the corresponding experiment specification of M12 (an adaptation of Figure 4.16(a)) was discarded. The remaining three experiment specifications were reused, i.e., the one with the property referring to LRP6 and CK1γ within lipid rafts (Figure 4.18(a) and Equation 4.5 and 4.6), the one with the property referring to nuclear β-catenin reaching a certain level in the later hours (an adaptation of Figure 4.16(b) and Equation 4.8),

and the one with the property referring to nuclear β-catenin under the condition that lipid rafts are removed (Figure 4.18(b) and Equation 4.9), with the expectation that none of those properties would be violated.

The simulation results of experimenting with M123 are presented in Figure 4.21, and all reused experiment specifications of M12 and M3 were checked successfully as expected, which shows the composed model preserves the key behavior from its model components as intended.

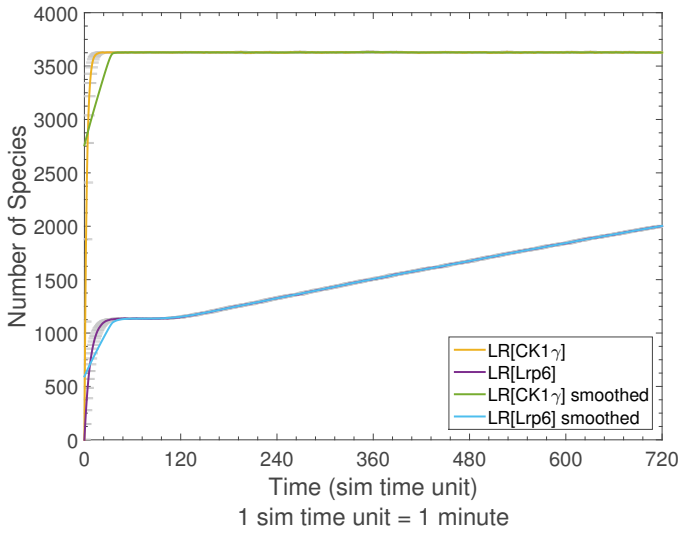
In addition, experiments were performed with the model M123 to check the property that model M12 is unable to produce, as shown in Equation 4.9, which has been the objective for the development of the model M123. The simulation results show that in the absence of lipid rafts the composed model M123 indeed reproduces an early peak of nuclear β-catenin, as presented in Figure 4.21(d).

Besides, further experiments showed that M123 is able to reproduce key observations from additional wet-lab experiments, e.g., when removing lipid rafts, a peak of nucleus β-catenin concentration at around 2 hours occurs, without the later increase [89]. To facilitate further reuse, model M123 has been annotated with all experiment specifications that were checked successfully.

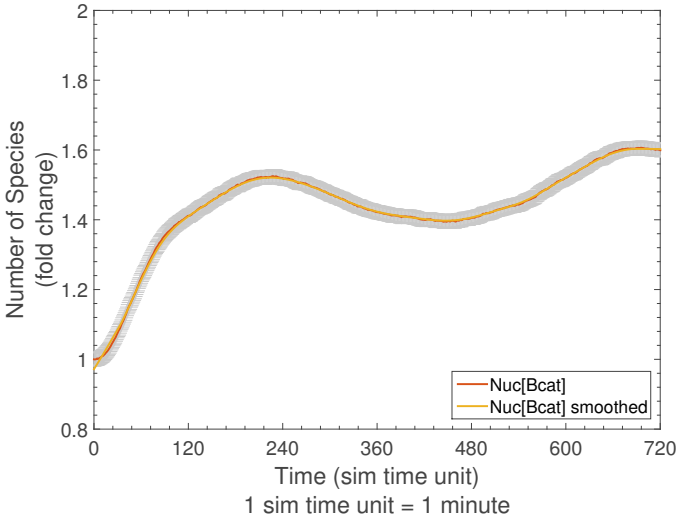
4.6. Summary

This chapter discussed reusing simulation experiments for developing models through model composition. Following a brief introduction on model composition and related work (Section 4.1), a life cycle was presented to structure the process of developing composed models, as well as different existing approaches for each phase of this process (Section 4.2). Based on this life cycle, the focus of this work was illustrated, which is to support semantic validation of the composed model. With the general approach having been presented in the previous chapter, the concept overview of reusing simulation experiments to support model composition was depicted in Section 4.3.1. Afterwards, problems in experiment adaptation that arise from this specific type of model reuse were discussed and two solutions were presented to tackle those problems (Section 4.3.2).

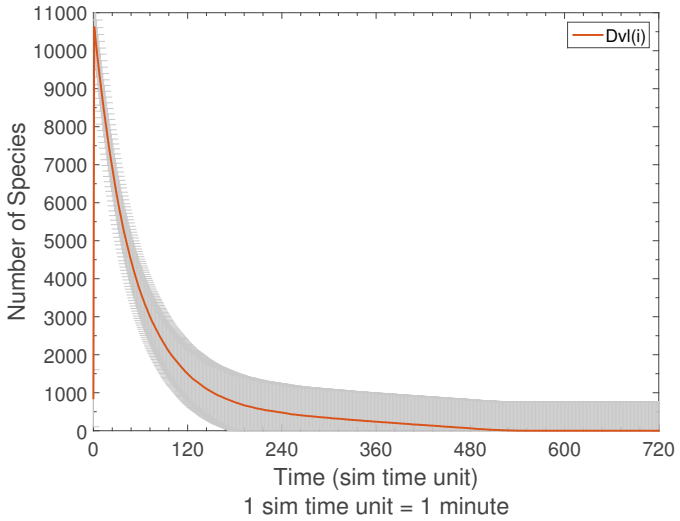
To demonstrate the applicability of the proposed approach, two case studies were presented, one based on composing variants of the small Lotka-Volterra model (Section 4.4) and one based on replaying the simulation study presented in [89] to develop a realistic model of the Wnt/β-catenin signaling pathway (Section 4.5). Whereas the case study of Lotka-Volterra models shows how the proposed approach can be used, the case study of the Wnt/β-catenin model illustrates the capability



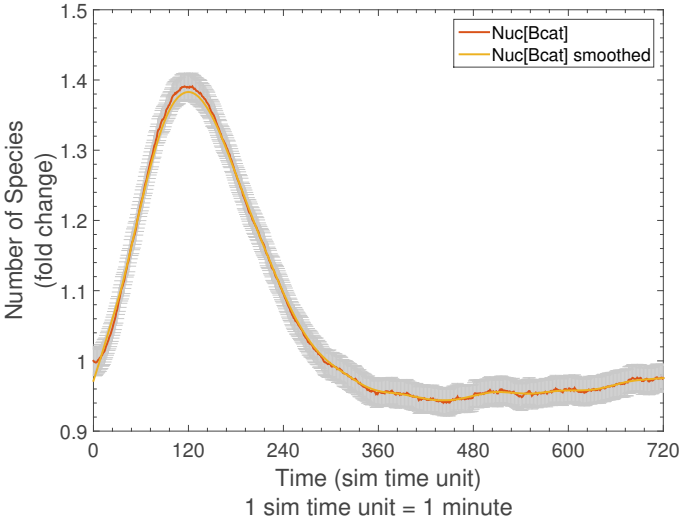
(a) The simulation results regarding Lrp6 and CK1γ.



(b) The simulation results regarding nuclear β-catenin with the default model configuration.



(c) The simulation results regarding Dvl.



(d) The simulation results regarding nuclear β-catenin when lipid rafts are removed, i.e., `nLR` = 0.

Figure 4.21.: Simulation results of the second composed model (M123), with the average trajectories with 95% confidence interval (gray error bars) and smoothed trajectories.

of the approach in supporting the development of complex models. Moreover, by following the life cycle of developing composed models, the second case study serves to give an idea of how the approach can be integrated with other numerous existing approaches in the area of model composition.

As shown by the case studies, simulation experiments of model components can be reused to automatically perform experiment with the composed model and check behavioral properties, to illuminate the impact of model composition on model behavior and ensure some key behavioral properties are still preserved after the composition. Also, it has been demonstrated that the proposed approach, together with the prototypical tool TAECS, can contribute to the semantic validation of the composed model and facilitate the complex modeling process of successive model composition.

Chapter 5

Model Extension Based on Reusing Simulation Experiments

In Chapter 3, the general concept of reusing simulation experiments to support model reuse has been presented, and in Chapter 4, the approach was applied to a specific type of model reuse, i.e., model composition, where different models are composed together to build larger ones. In this chapter, another type of model reuse is considered, i.e., model extension, where existing models are extended to build new ones, and the proposed approach will be applied to support developing models by extension. A case study based on developing a model of a receptor ligand pathway by successively extending existing ones, is employed to illustrate the process of model extension, and how the proposed approach can contribute to this process.

5.1. Model Reuse for Extension

While a substantial amount of work exists in facilitating model reuse, model composition has mainly been the focus of this research area, and reusing models for extension receives little attention. However, in many application domains, such as in the field of systems biology [72], acoustics [135] and electrical engineering [239], it is not uncommon to develop models based on extending existing ones, which is also termed as model refinement in some cases.

As illustrated by the three properties of models, i.e., the reference, the purpose, and the abstraction property (see Section 1.4, p. 9), every model is a simplification of the system under study for a specific purpose, and is valid with respect to this purpose. Therefore, any validated model can be further extended, e.g., to capture more phenomena of the modeled system, to represent the already described phenomena in more detail, or to tailor the model to a specific question.

Extending a model typically means adding more information so that the extended model is able to more accurately represent the modeled system. Thereby, model extension usually leads to increased complexity in the resulting model in comparison to the original one. However, building new models by extending existing ones allows dividing the complexity in modeling, as modelers can focus on specific questions of interest during the modeling and the remaining tasks can be achieved by reusing the original model. Thus, similar to model composition, model extension helps to reduce the required effort in the modeling process and facilitates the development of large and complex models. In model extension, models are built incrementally; once the model is validated regarding certain questions and purposes, new parts can be added. Those new parts usually are not sufficient to be abstracted into an independent, complete model, which is different from model composition, where individual models are composed together.

Similar to the life cycle of model composition shown in Figure 4.1, a life cycle of developing models by successive extension is presented in Figure 5.1, where five key steps are identified. On the one hand, model extension might ask more for exposing and changing the internal structure of the model than the black-box manner in model composition, and the creation of the extended model usually is accompanied by syntactic checking; on the other hand, syntactic verification is not the focus of this thesis. Therefore, in this life cycle, the syntactic verification step is implicitly incorporated in the model extension step. Like in the case of model composition, this thesis aims to support the validation of the extended model, as shown in Figure 5.1.

Some approaches have been developed to support model extension (refinement). For example, in [162] an approach is proposed to make refinement decision and answer the question "how much refinement of a simulation model is appropriate for a particular design problem [...]", by evaluating the refined model based on metrics such as economic payoff or utility. In [264], a procedure for refining models of gene-regulatory pathways is presented, which relies on the automated design of new wet-lab experiments with the modeled system using a score function, to obtain more experimental data so that existing models can be refined. Whereas those approaches are concerned with problems such as to what extent the model should be extended or how to derive the extended model, corresponding to the model extension step in the life cycle shown in Figure 5.1, this thesis focuses on validation of the extended model by employing simulation experiments.

As already stated, developing models is a process that is interwoven with performing simulation experiments. When models are extended, some key behavior of the original model should be preserved in the extended model and some should be

violated. Thus, it is of interest to know whether the extended model still behaves similarly to before and what are the changes in model behavior due to the extension. Simulation experiments are capable of providing information about model behavior; therefore, to analyze the impact of the extension on model behavior, often simulation experiments that have been conducted with the original model are re-conducted with the extended model and results are analyzed to compare behavior of the two models.

In this work, the idea of re-conducting simulation experiments of the original model with the extended model is made explicit and simulation experiments of individual models are reused to provide assistance in validation of the extended model.

5.2. Reusing Simulation Experiments for Model Extension

The overall concept of reusing simulation experiments for model extension is depicted in Figure 5.2. As illustrated in Chapter 3 and Chapter 4, the basis of the proposed approach is to annotate each model with specifications of simulation experiments that have been executed with it. In the experiment specification, the expected model behavioral properties are explicitly defined along with the experiment conditions

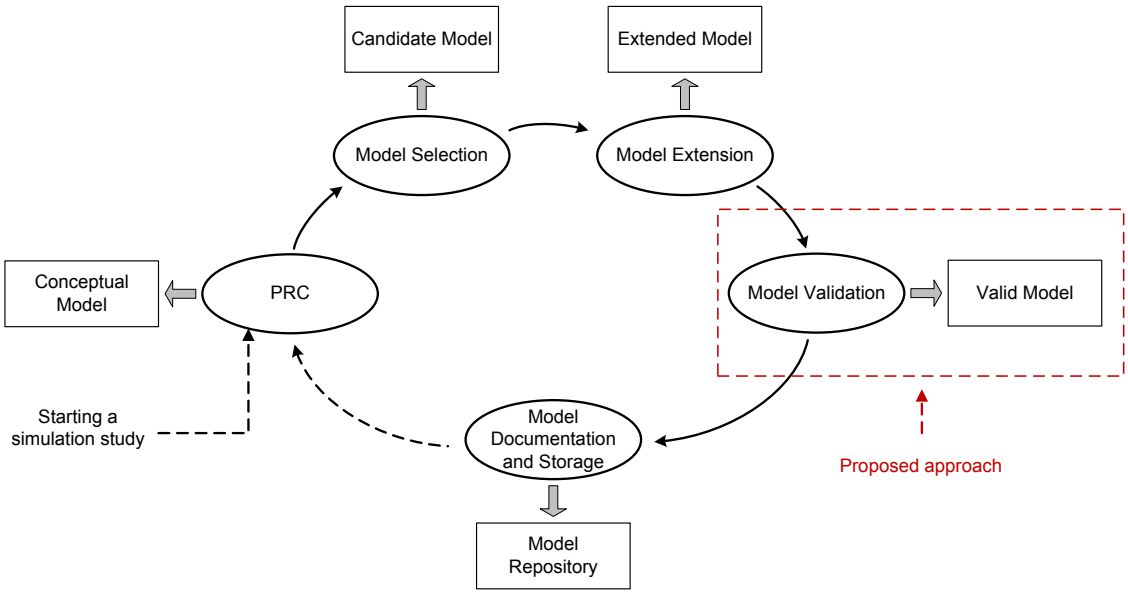


Figure 5.1.: A life cycle of modeling based on extension. PRC stands for Problem Formulation, Requirements Engineering and Conceptual Modeling, which are merged into one phase in this life cycle with a conceptual model developed. Usually a simulation study starts with the PRC phase.

under which the model's behavior can satisfy those properties, corresponding to step 1 in Figure 5.2. The user can take an existing model together with its experiment specifications, and extend the model, corresponding to step 2. After the model extension, an extended model is created; also, adaptation information that contains changes regarding model variables and their configuration is derived when necessary, and result expectation referring to checking model behavioral properties on the extended model is specified as well.

Subsequently, based on possible adaptation information, experiment specifications of the original model are reused, adapted, and executed for the extended model with the tool TAECS. Experiment results are analyzed to check whether the specified model behavioral properties are satisfied or not, and whether this is as expected. This procedure corresponds to the step 3 in Figure 5.2.

Apart from reusing experiment specifications of the original model, users can additionally design and perform simulation experiments with the extended model, and the specifications of those experiments are executed and checked as well, corresponding to step 4. When an experiment specification is reused and executed, and the generated experiment results cannot be checked successfully against the specified behavioral properties, it is returned to the user, which can be revised and then checked again, corresponding to step 5. Each experiment specification that has been checked successfully is added as an annotation to the extended model, as shown in step 6.

As discussed in the previous two chapters, the experiment specification relies on SESSL, and the developed tool TAECS is used for automatic adaptation and execution of simulation experiments as well as result analysis. Experiment adaptation for model extension is more straightforward than that for model composition, as only one model is reused. During the extension, when model variables are renamed or configuration of model parameters is changed, this information should be included in the adaptation information, either through user specification as the case in the current implementation, or through automated derivation based on model transformation or model version control (see discussion in Section 3.5.3, pp. 61 - 62). When new model parameters are introduced, their configuration can be defined in the model file or included as part of the adaptation information. In either case, a complete default model configuration should be provided in the model file. The whole procedure of adapting, executing, and checking experiments is as presented in Algorithm 1.

By reusing simulation experiments of the original model, automatically performing experiments with the extended model, and checking the behavior of the extended model against the properties that are satisfied by the original model, the impact of model extension on the model behavior can be illuminated and some insights can be

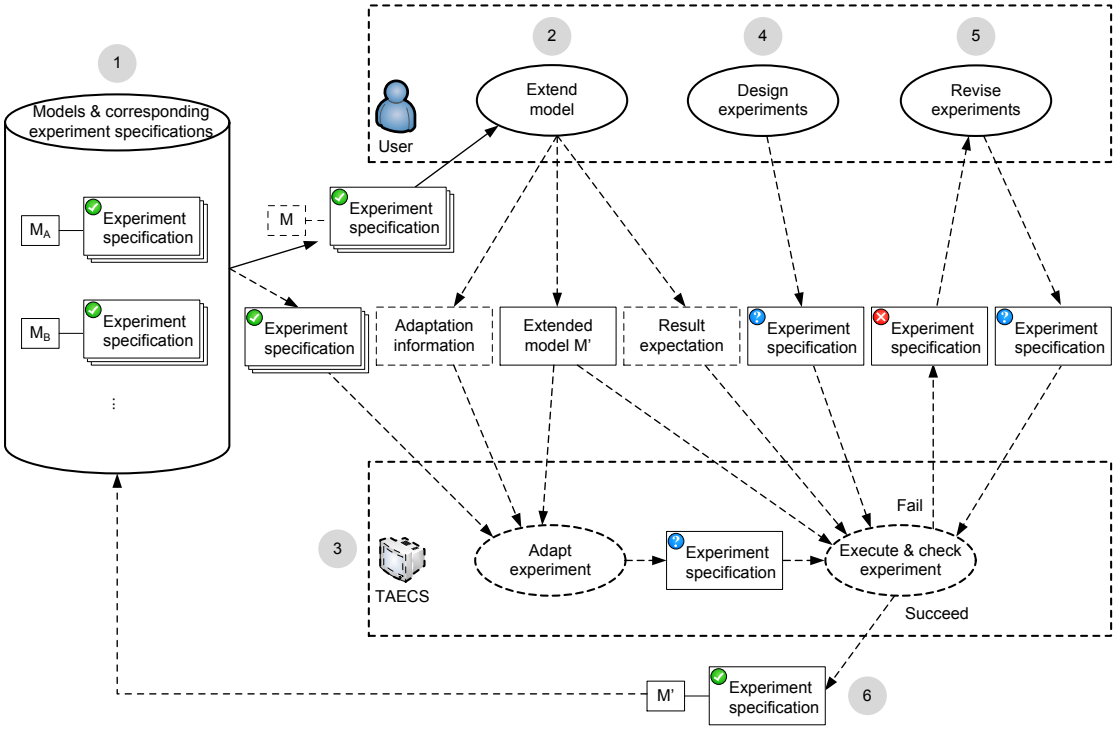


Figure 5.2.: The overview of reusing simulation experiments for model extension. When the user extends an existing model, the corresponding experiment specifications of the original model, adaptation information, result expectation and the extended model are passed to TAECS, which first adapts the experiments and then executes and checks them. Besides, TAECS can directly execute and check experiment specifications, which are created by the user either through performing new experiments or through revising experiments that fail in previous checking. When checked successfully, the model is annotated with the experiment specification.[Adapted from [186]]

gained on the validity of the extended model. When models are successively extended, the proposed approach can assist in ensuring that some key characteristics are not disrupted, and can provide fast feedback to the modeler during model extension.

5.3. Case Study III: Modeling the Receptor Ligand Pathway

In this case study, the proposed approach was applied to modeling the process of receptor ligand binding and subsequent signaling events, as shown in Figure 5.3. Three models at various levels of detail were developed by successive extension. First, a basic model was developed to describe the formation of a simple receptor ligand

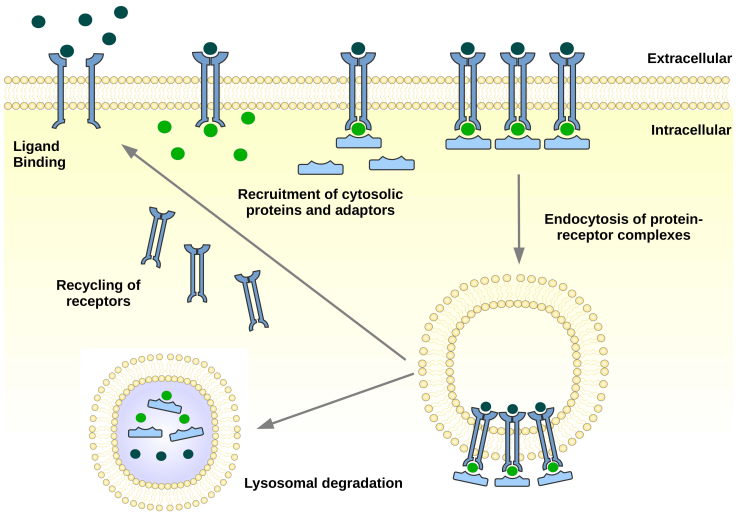


Figure 5.3.: The receptor ligand pathway. [From [186]]

binding at the membrane. Subsequently, this basic model was successively extended by adding two additional processes into the model one after another.

5.3.1. Background

The binding of a ligand to a membrane-integral receptor plays a pivotal role in cell communication. It initiates the cellular signal transduction and provides the opportunity to transmit signals and information from the outside of the cell into the inside. Once a ligand-receptor complex is successfully established, an intracellular signaling cascade is initiated, which eventually leads to the input-specific response of the cell, such as the specific expression of a target gene. The strength/effectiveness of the signal response depends, among others, on how many ligands can be bound and, in particular, how stable the complex is, i.e., how long the binding between receptor and ligand lasts.

The stability as well as the function of the receptor complex is greatly effected by other membrane-associated (peripheral) proteins that interact with the ligand-receptor complex. Therefore, the interaction may eventually lead to the formation of a ternary complex, which consists of the ligand, the receptor and the peripheral protein. Common examples for proteins involved in the formation of ternary complexes are G-proteins, coated pit adaptors, cytoskeletal elements or other receptors [123]. The impact of such receptor coupling interactions and the formation of a ternary complex is two-fold. On the one hand, the association and dissociation rates for receptor-ligand binding can vary significantly between binary and ternary complexes —a fact that has to be taken into close consideration when analyzing experimental data. Renown

examples for this effect are receptor/G-protein coupling and EGF receptor/adaptor coupling. On the other hand, the interaction with membrane-associated proteins induces further signaling events, like internalization (endocytosis) and recycling processes or receptor accumulation.

Therefore, this case study aimed to capture the essential receptor-ligand binding kinetics in dependence of the previously described receptor mediated events, i.e., receptor-protein coupling, internalization and recycling.

5.3.2. Ingredients

As in previously presented case studies, the multi-level rule-based modeling language ML-Rules (see Section 4.4.2 for more details, pp. 87-88) was chosen to describe models in this case study. To perform simulation experiments, the modeling and simulation framework JAMES II was employed to execute the experiments specified in SESSL, via the binding `sessl.james`. Within the experiments, the simulation algorithm described in [153, 261] was used, named as `MLRulesReference` in SESSL.

To apply the proposed approach during the model extension, experiments of individual models were specified in SESSL. As demonstrated by the case study of the Wnt/β-catenin pathway, $\text{MITL}_{[a,b]}$ has been shown to be a suitable method for describing model behavioral properties, as it allows the expression of property qualitatively based on traditional operators in LTL and further supports explicitly relating properties with time intervals. Also, the method described in Section 4.5.2 (pp. 104-105) was used for checking properties expressed in $\text{MITL}_{[a,b]}$.

As ML-Rules is based on Continuous-Time Markov Chain semantics, stochasticity needs be to considered in performing simulation experiments, property specification, and result analysis. Thus, multiple replications are required. Resembling the case study of Lotka-Volterra models, this case study also employed CSL to express probability regarding replications (Section 4.4.2, pp. 88-89), and probabilistic statements can be defined as $Pr_{\bowtie p}(\phi)$, where $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0,1]$, and ϕ is an $\text{MITL}_{[a,b]}$ formula. Besides, the method for statistical model checking presented in Section 4.4.2 (pp. 88-89), which is based on hypothesis testing, was used to check whether the probability of the given property holding for a randomly selected simulation run satisfies the specified requirement. The method presented in Algorithm 2 was used to determine the minimal needed replications, as discussed in Section 4.4.2 (pp. 89-90); parameters for this algorithm were configured as $\alpha = \beta = \delta = 0.05$ throughout this case study. To alleviate the stochasticity within simulation trajectories of one replication, as in the case study of the Wnt/β-catenin pathway (Section 4.5.2, p. 104), the LOESS smoothing method was exploited to pre-process

raw simulation data, with the configuration of the bandwidth parameter being 0.1 and the number of robustness iterations parameter being 0.

Furthermore, the implementation of the proposed approach TAECS was used to automatically generate simulation experiments by reusing the original ones, execute these adapted experiments, and perform statistical model checking on the experiment results.

5.3.3. The Basic Model — Ternary-Complex Formation (M0)

According to the life cycle of developing models by extension presented in Figure 5.1, firstly a conceptual model should be established, based on which a candidate model is selected for the model extension. Here, the process of conceptual modeling is omitted, as this is not the focus of this thesis. Also, this case study started with the development of the ternary-complex model (M0), which forms the basis for subsequent model extension.

As the most basic version, this model contains four different species: ligand (L), receptor (R), adaptor protein (X) and a representation for protein complexes (C), which can have different states, depending on how many compounds are bound. As shown in Figure 5.4, two scenarios exist regarding the formation process of the ternary-complex: one is that the ligand binds the receptor first and then cytosolic proteins binds the ligand-receptor complex (depicted with solid lines in the figure), and another is that cytosolic proteins bind the receptor first and afterwards the ligand is bound (depicted with dashed lines in the figure).

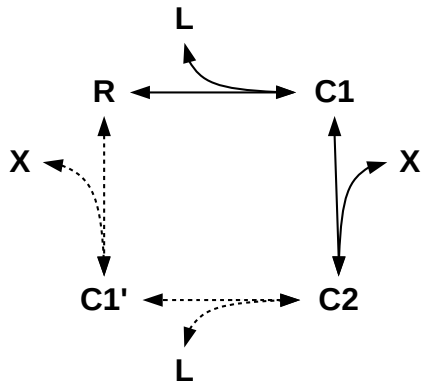


Figure 5.4.: The two scenarios of the ternary-complex formation.

For simplicity, the second scenario, where the binding of cytosolic proteins and the receptor is formed first, was disregarded, based on the fact that reaction rate constants for this scenario are significantly lower, and thus can reasonably be neglected in the

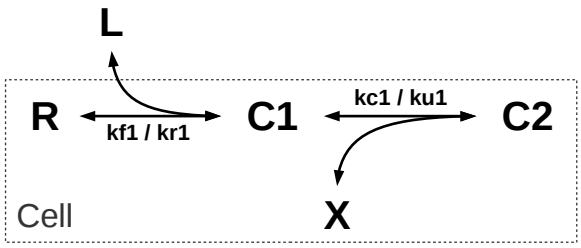


Figure 5.5.: The ternary-complex model (M0). [From [186]]

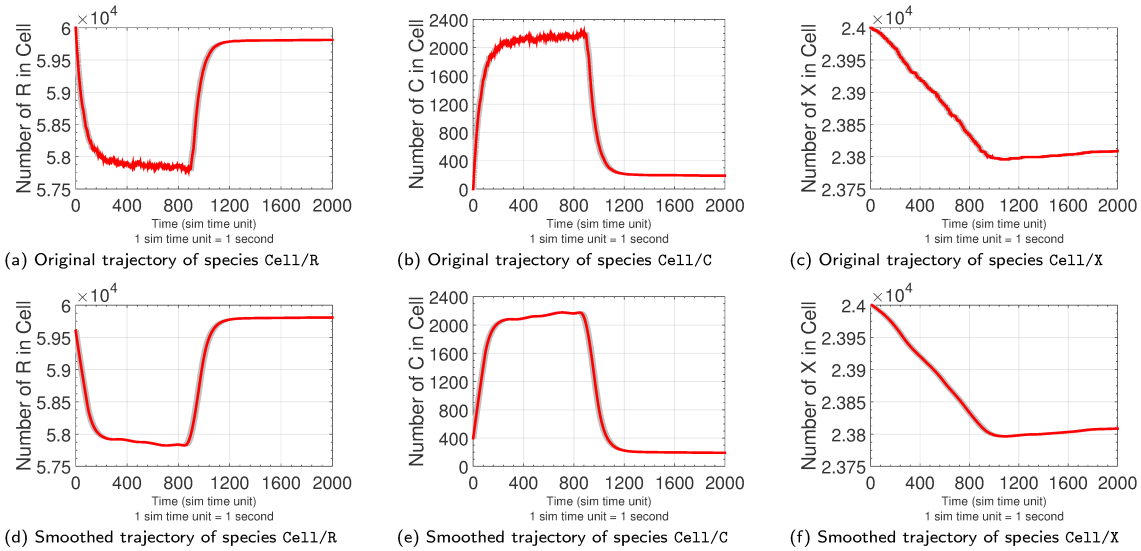
model. Therefore, in the basic model, the first scenario was captured, where the formation of the ternary-complex starts with the ligand bound to the receptor, as depicted in Figure 5.5.

The ternary-complex model is of general nature, but can be parametrized and fitted to match the dynamics of specific pathways. This case study aimed to reproduce EGF / EGFR binding kinetics that had been experimentally determined in fibroblasts. In the wet-lab experiments, cells were stimulated for a certain time with EGF, and the binding kinetics were observed by measuring the amount of ligand/receptor complexes at different time points. After a given time, cells were washed and EGF was completely removed from the system. For this system, all reaction rate constants are available and were used to parametrize the ternary-complex model (see supplementary file in [186]). In the model, a concentration of 5×10^{-10} was applied as EGF stimulus for a time period of 15 minutes, with one second corresponding to one simulation time unit. All remaining EGF molecules are removed after 15 minutes (i.e., 900 simulation time units), which is similar to the wet-lab experimental set-up [154].

To inspect the behavioral properties of the ternary-complex model, the number of receptor (R), adaptor protein (X) and the number of a representation for protein complexes (C) were selected as species of interest. Let `nR, nL` and `nX` denote the initial number of R, L and X, respectively, and model parameters were configured as follows:

```
nR: 6e04,  nL: 1.05e09,  nX: 2.4e04,
```

To perform simulation experiments for exploring model behavior, initially the simulation stop time was set to 2000 simulation time units and the replication number for each simulation experiment to 200. During the simulation, the number of the three species (R, C and X) were observed, denoted as variable `Cell/R`, `Cell/C` and `Cell/X` in the model, respectively. The simulation trajectories of the observed variables are shown in Figure 5.6 (a), (b) and (c), which are in good agreement with wet-lab experimental results [154]. Due to the stochasticity, some noise exists in the



(a) Original trajectory of species `Cell/R` (b) Original trajectory of species `Cell/C` (c) Original trajectory of species `Cell/X`

(d) Smoothed trajectory of species `Cell/R` (e) Smoothed trajectory of species `Cell/C` (f) Smoothed trajectory of species `Cell/X`

Figure 5.6.: Simulation results of the ternary-complex model (M0). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b) and (c) are the original trajectories from simulation experiments; (d), (e) and (f) are the smoothed trajectories using the LOESS method on the original ones, respectively. Please note, as a result of the smoothing the initial values of trajectories (d) and (e) slightly deviate from the original trajectories (a) and (b). However, this deviation does not affect the considered model behavioral properties, hence does not bias test results. [From [186]]

generated simulation trajectories. To diminish the impact of stochastic fluctuations in those trajectories, the LOESS method was applied, and the trajectories after smoothing are shown in Figure 5.6 (d), (e) and (f). The LOESS method may influence the initial values in the trajectories, such as decreasing or increasing the values, e.g., in Figure 5.6 (d) and (a), or (e) and (b). However, this decreasing or increasing of values does not impact the model behavioral properties considered in this case study. This also applies to the experiments of the two extended models ME1 and ME2, which will be discussed in Section 5.3.4 and 5.3.7, i.e., Figure 5.9 and 5.13.

According to the simulation results, three model behavioral properties were derived by the user, each referring to one species of interest, described as follows:

- Species `Cell/R`: with a probability of no less than 0.75, the species number remains within the range [57000, 58000] over the time interval [300, 800], and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [59750, 59850].

- Species `Cell/C`: with a probability of no less than 0.95, the species number remains within the range [2000, 2300] over the time interval [300, 800], and reaches a steady state over the time interval [1300, 2000] with the amount being in the range [150, 250].
- Species `Cell/X`: with a probability of no less than 0.85, the species number decreases over the time interval [0, 800] and increases over the time interval [1300, 2000].

Based on the simulation set-up and derived properties, the experiments were specified in SESSL into three specifications, each referring to one species of interest. The three experiment specifications are presented in Figure 5.7. All three experiment specifications have the same model configuration, simulation configuration and data processing method, but different model behavioral properties. The complete SESSL experiment specification regarding species `Cell/R` is shown in Figure 5.7(a), including the model configuration, simulation configuration, data processing method and model behavioral property. As for experiments regarding species `Cell/C` and `Cell/X`, only the property specifications are depicted, for simplicity, in Figure 5.7(b) and 5.7(c), respectively.

To describe properties that a variable increases or decreases, the first order derivative of the variable is used, specified as `d("variableName")` in SESSL. For example, a property that the number of species `Cell/X` decreases is expressed as `d("Cell/X") < 0` in SESSL, standing for the first order derivative of the number of `Cell/X` is less than 0, as shown in Figure 5.7(c). For each variable, the calculation of its first order derivative is based on the variable value at observed time points and observation interval. In the prototypical implementation, the first order derivative value of a variable at time point t_i is calculated as the slope to the variable value at next observed time point t_{i+1}. If no next time point exists, the derivation value from previous time point is used.

Using the tool TAECS, each of the three experiment specifications was tested on the ternary-complex model (M0). The calculated minimal replication numbers needed for checking the properties of `Cell/R`, `Cell/C` and `Cell/X` are 205, 77 and 154, respectively. All experiment specifications were checked successfully, and therefore the ternary-complex model M0 was annotated with the three experiment specifications.

5.3.4. Model Extension — Endocytosis (ME1)

The binding of the ligand-receptor complex to adaptor proteins (such as clathrin) often induces the endocytosis of the complex, which means the receptor complex is

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with Hypothesis {
  model = "file-mlrj:/./TernaryComplexModel.mlrj"
  set("nR" <~ 6e04, "nL" <~ 1.05e09, "nX" <~ 2.4e04)
  simulator = MLRulesReference()
  stopTime = 2000
  observe("Cell/R")
  observeAt(range(0, 1, 2000))
  dataPreProcessor = LoessInterpolation(0.1, 0)
  assume{(Probability >= 0.75)(
   G(300, 800, (variable("Cell/R") >= 57000) and (variable("Cell/R") <= 58000))
    and G(1200, 2000, ((variable("Cell/R") >= 59750) and variable("Cell/R") <= 59850))
    )}
  }
```

(a) Experiment specification regarding species `Cell/R`.

```
assume{(Probability >= 0.95)(
 G(300, 800, variable("Cell/C") >= 2000 and variable("Cell/C") <= 2300)
  and G(1300, 2000, variable("Cell/C") >= 150 and variable("Cell/C") <= 250)
 )}
```

(b) Specification of model behavioral property regarding species `Cell/C`.

```
assume{(Probability >= 0.85)(
 G(0, 800, d("Cell/X") < 0) and G(1300, 2000,d("Cell/X") > 0)
 )}
```

(c) Specification of model behavioral property regarding species `Cell/X`. `d("Cell/X") < 0` (`d("Cell/X") > 0`) means the first order derivative of the number of `Cell/X` is less (more) than 0, indicating that the number of `Cell/X` decreases (increases).

Figure 5.7.: Experiment specifications of the ternary-complex model (M0) in SESSL. [Adapted from [186]]

internalized into the cytosol as part of an endosome. As a result, ligand, receptor and adaptor proteins are not available at the cell surface anymore.

As the ternary-complex model (M0) describes the formation of the ternary-complex containing the ligand, the receptor, and adaptor proteins, this basic model was selected for extension to capture the process of endocytosis. Hence, when extending the ternary-complex model, all of the existing rules, parameters, and their configuration remained the same. Additionally, a new species endosome was added, and a first order reaction was introduced, which describes the internalization of the ternary complex into a cytoplasmic endosome with a certain rate constant $ke1$, as shown in Figure 5.8. The configuration of new parameters was based on literature

(see supplementary file from [186]). No renaming of model variables was involved and the configuration of all parameters were defined in the model file.

This model extension would have an impact on the observed behavior properties from the ternary-complex model. On the one hand, the endocytosis leads to a portion of ternary-complexes encapsulated into endosomes, and therefore the location of those species are changed, i.e., `Cell/C` → `Cell/Endosome/C`. Consequently, it was assumed that the maximum amount of `Cell/C` would be lower in the endocytosis model (ME1) than in the ternary-complex model (M0), as some of the complexes C are also located within endosomes (`Cell/Endosome/C`). Therefore, the properties defined and checked for `Cell/C` in the ternary-complex model were expected to be violated for the endocytosis model. On the other hand, in the extended model ME1, once the ternary-complex is encapsulated into the endosome, they are not transferred out. As a result, species that make up of those complexes, i.e., `Cell/R` and `Cell/X`, are not available for the backward reactions (*kr1, ku1*), and hence their amount should decrease by the number of complexes internalized. Thus, the property regarding the two species should not hold in the endocytosis model (ME1) as well.

5.3.5. Reusing Simulation Experiments of the Ternary-Complex Model

To investigate the behavior of the extended model ME1 and gain information on its validity, the proposed approach was applied and the simulation experiments associated with the ternary-complex model (M0) were reused. As during the model extension no changes in model parameters were defined in the adaptation information, in the reused experiment specifications only the model location needed to be updated from the original model (M0) to the extended model (ME1).

With the extended model ME1, three experiment specifications depicted in Figure 5.7, the adaptation information containing the new model location, and the result

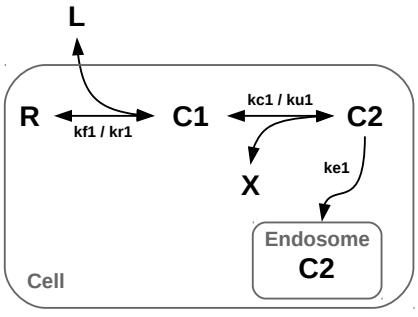


Figure 5.8.: The endocytosis model (ME1). [From [186]]

expectation that all the three experiments would fail, the tool TAECS performed tests on the extended model ME1. Experiments were automatically generated and executed with the model ME1, and simulation results were checked against the specified properties. Table 5.1 presents the testing result of the three reused experiment specifications, for each of which it includes the calculated minimal replication number for statistical model checking, the expected checking result of the specified property, and the actual checking result. The simulation results are shown in Figure 5.9, while the raw trajectories regarding the three species `Cell/R`, `Cell/C`, and `Cell/X` are depicted in (a), (b) and (c), their smoothed trajectories are presented in (e), (f), (g).

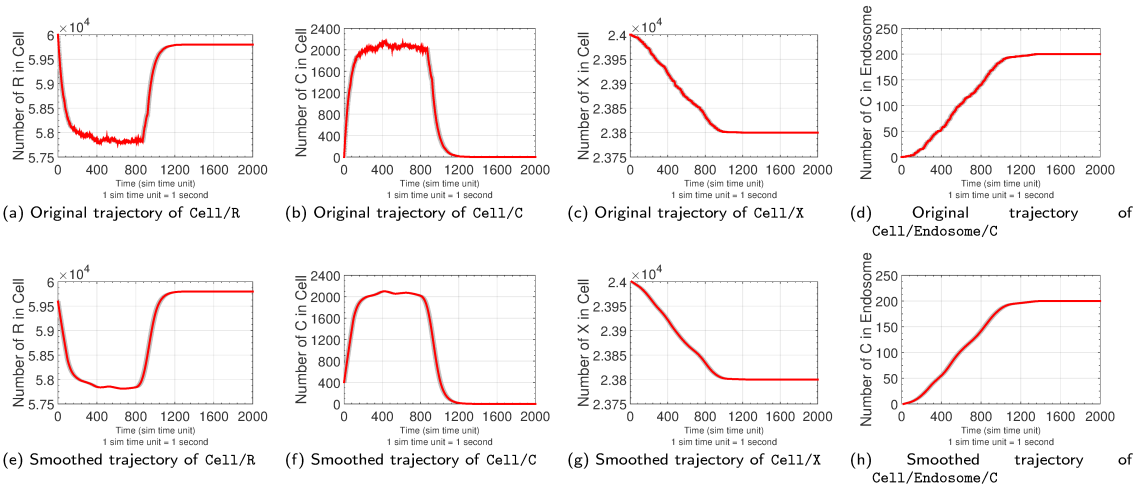
According to the discussion in Section 5.3.4 about the influence of model extension on the model behavior, the three behavioral properties from the ternary-complex model should all be violated in the endocytosis model ME1. However, the testing results revealed that only the properties regarding `Cell/C` and `Cell/X` are violated, whereas the property regarding `Cell/R` still holds in the endocytosis model, which is contradictory to the expectation.

By closely checking the simulation experiments, it was found that this contradiction results from insufficient simulation run time and can be resolved by executing the simulation for longer time. As a matter of fact, in the simulation experiments with the ternary-complex model M0, the number of species `Cell/R` does not end up with a steady state after the washing time (900 simulation time units and 15 minutes in wet-lab experiments, see discussion in Section 5.3.3). Instead, the species number increases but slowly so that when the simulation stops after 2,000 time units, this increasing is not noticeable, as shown in Figure 5.6 (a). With a sufficiently long simulation run time, e.g., 30,000 rather than 2,000 time units, the increasing in the species number becomes distinct, as shown in Figure 5.10 (a).

However, in the simulation experiment with the endocytosis model ME1, when the simulation run time is configured as 30,000 units, the number of species `Cell/R` reaches a steady state from a certain time after the washing step, with the species

Property	Replication number	Assumed result	Checking result
Cell/R	205	false	true
Cell/C	77	false	false
Cell/X	154	false	false

Table 5.1.: Results of testing on the endocytosis model (ME1) by reusing experiments of the ternary-complex model (M0). [From [186]]



(a) Original trajectory of `Cell/R` (b) Original trajectory of `Cell/C` (c) Original trajectory of `Cell/X` (d) Original trajectory of `Cell/Endosome/C`

(e) Smoothed trajectory of `Cell/R` (f) Smoothed trajectory of `Cell/C` (g) Smoothed trajectory of `Cell/X` (h) Smoothed trajectory of `Cell/Endosome/C`

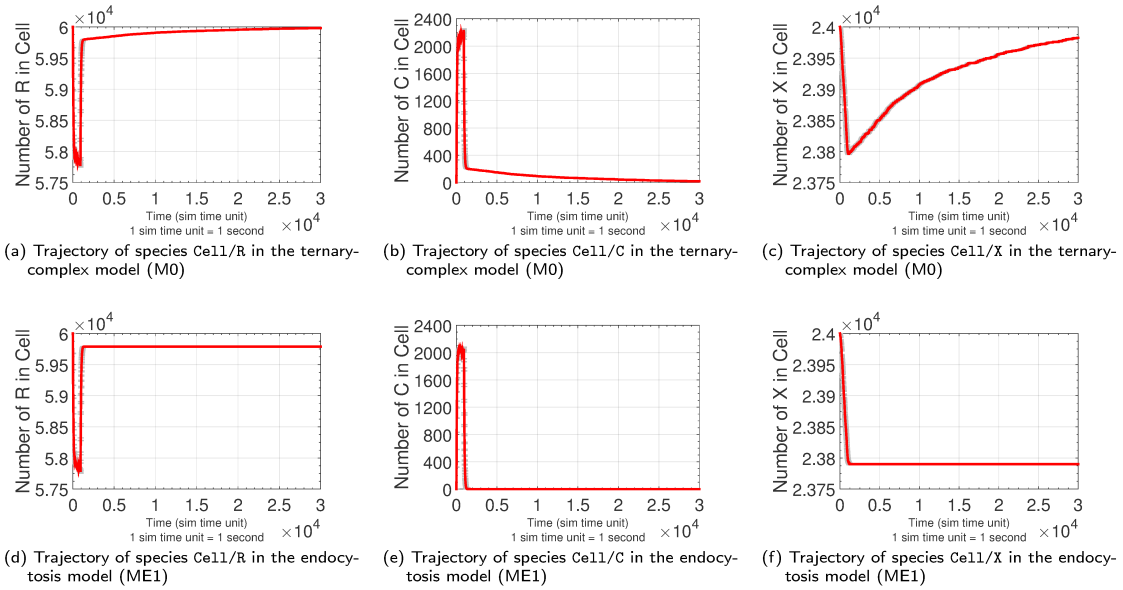
Figure 5.9.: Simulation results of the endocytosis model (ME1). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b), (c) and (d) are the original trajectories from simulation experiments; (e), (f), (g) and (h) are the smoothed trajectories using the LOESS method on the original ones, respectively. [From [186]]

number being lower than that for the ternary-complex model, as shown in Figure 5.10 (d). Thus, the extended model ME1 indeed exhibits a different behavior from the original ternary-complex model M0, not only regarding the two species `Cell/C` and `Cell/X`, but also the species `Cell/R`. This is in agreement with expectation, and shows that the extension with the endocytosis process does work as intended.

5.3.6. Model Documentation

From the simulation trajectories shown in Figure 5.10 (d), (e) and (f), one can see that in experiments with the endocytosis model ME1, there is no significant change in the trajectories after 2,000 time units and the model behavior reaches steady state. As experiments with simulation run time being 2,000 units are able to reflect the main behavioral properties of the extended model ME1, for better illustration this case study stayed with the configuration of simulation stop time being 2,000 time units for the subsequent simulation experiments.

The experiment specification from the basic model M0 with the property regarding the species `Cell/R` was checked successfully on the extended model ME1 after adaptation. Thus, the adapted experiment specification, as described in Figure 5.11(a), was annotated together with the model ME1. However, the other two experiment specifications of the ternary-complex model (M0), i.e., regarding species



(a) Trajectory of species `Cell/R` in the ternary-complex model (M0)
(b) Trajectory of species `Cell/C` in the ternary-complex model (M0)
(c) Trajectory of species `Cell/X` in the ternary-complex model (M0)
(d) Trajectory of species `Cell/R` in the endocytosis model (ME1)
(e) Trajectory of species `Cell/C` in the endocytosis model (ME1)
(f) Trajectory of species `Cell/X` in the endocytosis model (ME1)

Figure 5.10.: Simulation trajectories of the ternary-complex model (M0) and the endocytosis model (ME1) with long simulation run time (30,000 time units). The mean trajectories (in red) with the standard error (in gray error bars) are shown. (a), (b) and (c) are the trajectories from the ternary-complex model; (d), (e) and (f) are the trajectories from the endocytosis model. [From [186]]

`Cell/C` and `Cell/X` as depicted in Figure 5.7(b) and 5.7(c), failed in the checking on the model ME1, and therefore required revision. Based on the simulation results shown in Figure5.9(b), (c), (f), and (g), the properties referring to the species `Cell/C` and `Cell/X` were updated.

In addition, based on how the original model was extended, another species is of interest for the endocytosis model ME1, namely the receptor complex in the endocytosis denoted as `Cell/Endosome/C`. Therefore, additional simulation experiments were performed with the model ME1, to observe the species `Cell/Endosome/C`. The simulation trajectories are shown in Figure5.9 (d) and (h), based on which the property regarding the species `Cell/Endosome/C` was acquired.

The two updated properties and one newly derived property for the model ME1 are described as below:

- Species `Cell/C`: with a probability of no less than 0.75, the species number reaches a steady state over the time interval [300, 800] with the amount being in the range [1900, 2200], and also reaches a steady state over the time interval [1600, 2000] with the amount being 0.

- Species `Cell/X`: with a probability of no less than 0.75, the species number decreases over the time interval [0, 900] and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [23700, 23800].
- Species `Cell/Endosome/C`: with a probability of no less than 0.75, the species number increases over the time interval [0, 900] and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [200, 250].

For all of the three properties above, in their corresponding simulation experiment the model configuration, the simulation configuration and the data processing method are the same as that in the experiment specification regarding species `Cell/R`, as shown in Figure 5.11(a). The three experiments were specified in SESSL as well, and the property specifications are represented in Figure 5.11(b), 5.11(c), and 5.11(d).

Furthermore, all the four experiment specifications were checked with TAECS using the procedure of case B shown in Figure 3.9, each of which succeeded. Hence, the endocytosis model (ME1) was annotated with four simulation experiment specifications, i.e., experiments regarding species `Cell/R`, `Cell/C`, `Cell/X` and `Cell/Endosome/C`, as depicted in Figure 5.11.

5.3.7. Further Model extension — Recycling (ME2)

When the receptor complex is encapsulated into endocytosis, typically those internalized complexes are dissociated. Subsequently, the dissociated species are recycled: the receptor is returned to the membrane, while the ligand and adaptor protein are transferred to a lysosome, where they are degraded, as shown in Figure 5.12.

To include this recycling process, the endocytosis model ME1 was further extended, leading to the development of the second extended model, i.e., the recycling model ME2. All the parameters and reactions from the model ME1 were reused, and one new reaction was added, which describes the recycling of the receptor from the dissociated ternary-complex into the membrane, i.e., `Cell/Endosome/C` $\rightarrow$ `Cell/R`. During the extension, no change had been made in the model configuration from the model ME1 and the new parameter was configured in the model file of ME2.

Due to the introducing of the recycling process, in the second extended model ME2, the concentration of the receptors in the cell should be gradually restored after the washing step, resembling the dynamics of the ternary-complex model M0, but not the endocytosis model ME1. Inside the cell, the ternary complex and its remaining components, i.e., `Cell/C` and `Cell/X`, are still partially encapsulated into the endosome without a backward reaction; therefore, the dynamics of the two species are not influenced by the recycling, and do not change in comparison

```
val exp = new Experiment with Observation with Hypothesis {
  model = "file-mlrj:/./EndocytosisModel.mlrj"
  set("nR" <~ 6e04, "nL" <~ 1.05e09, "nX" <~ 2.4e04)
  simulator = MLRulesReference()
  stopTime = 2000
  observe("Cell/R")
  observeAt(range(0, 1, 2000))
  dataPreProcessor = LoessInterpolation(0.1, 0)
  assume{(Probability >= 0.75)(
   G(300, 800, (variable("Cell/R") >= 57000) and (variable("Cell/R") <= 58000))
    and G(1200, 2000, ((variable("Cell/R") >= 59750) and variable("Cell/R") <= 59850))
  )}
}
```

(a) Experiment specification regarding species `Cell/R`.

```
assume{(Probability >= 0.75)(
 G(300, 800, variable("Cell/C") >= 1900 and variable("Cell/C") <= 2200)
  and G(1600, 2000, variable("Cell/C") == 0)
)}
```

(b) Specification of model behavioral property regarding species `Cell/C`.

```
assume{(Probability >= 0.75)(
 G(0.0, 900, d("Cell/X") <= 0) and
  G(1200, 2000, variable("Cell/X") >= 23700 and variable("Cell/X") <= 23800)
)}
```

(c) Specification of model behavioral property regarding species `Cell/X`.

```
assume{(Probability >= 0.75)(
 G(0, 900, d("Cell/Endosome/C") > 0) and
   G(1200, 2000, variable("Cell/Endosome/C") >= 200 and variable("Cell/Endosome/C") <= 250)
)}
```

(d) Specification of model behavioral property regarding species `Cell/Endosome/C`.

Figure 5.11.: Experiment specifications of the endocytosis model (ME1) in SESSL. [Adapted from [186]]

to the endocytosis model. In contrast, the species C located inside the endosome (`Cell/Endosome/C`) is being degraded, and thus cannot stay aggregated in the endosome as in the endocytosis model ME1. Hence, when checking the behavioral properties of the endocytosis model ME1 on the recycling model ME2, the expectation of the checking result was that whereas the properties regarding species X and species C in Cell still hold, the properties regarding species `Cell/R` and `Cell/Endosome/C` do not.

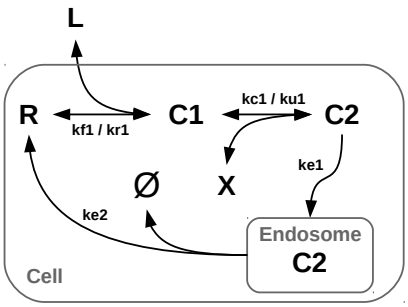


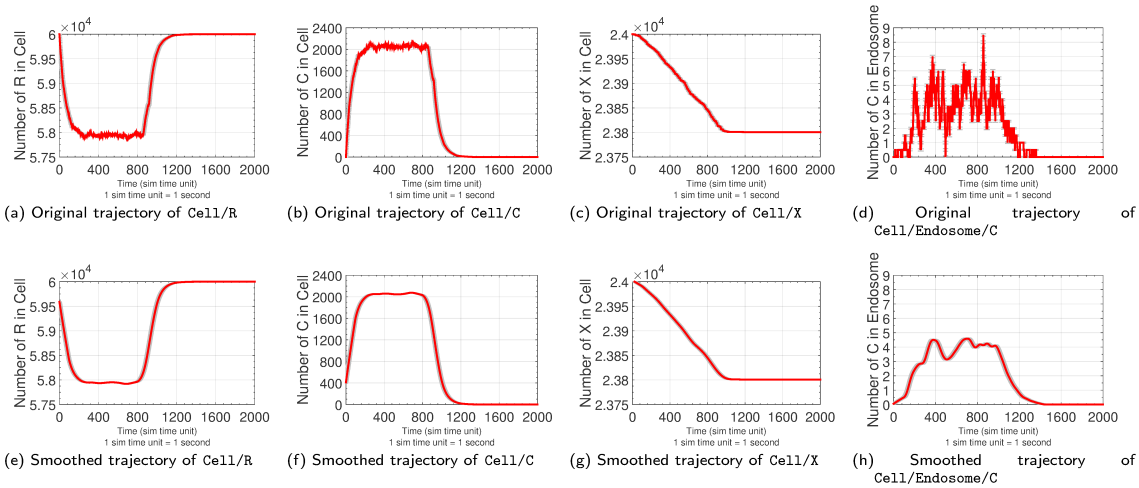
Figure 5.12.: The recycling model (ME2). [From [186]]

The proposed approach was applied to test the recycling model ME2, and the four experiment specifications of the endocytosis model ME1 depicted in Figure 5.11 were reused. No adaptation for model parameters and properties in those experiment specifications was needed, and only the model location was updated to the model ME2 from the reused model ME1. Using the procedure of the case A presented in Figure 3.9, the tool TAECS performed simulation experiments with the model ME2, by adapting the experiment specifications of ME1, and experiments results were evaluated against specified properties.

Test results are presented in Table 5.2, which show that the experiments regarding species `Cell/C` and `Cell/X` were checked successfully, while the experiments regarding species `Cell/R` and `Cell/Endsome/C` failed. The checking results are in agreement with the expectation, and thus the overall test result is true, indicating that the model extension of recycling worked as intended. Simulation trajectories from the experiments are shown in Figure 5.13. In order to facilitate further reuse, the model ME2 should be annotated with the experiment specifications that were successfully checked, i.e., those regarding species `Cell/C` and `Cell/X`, whereas the failed ones need revision.

Property	Replication number	Expected result	Checking result
Cell/R	205	false	false
Cell/C	205	true	true
Cell/X	205	true	true
Cell/Endosome/C	205	false	false

Table 5.2.: Results of testing on the recycling model (ME2) by reusing experiments of the endocytosis model (ME1). [From [186]]



(a) Original trajectory of `Cell/R` (b) Original trajectory of `Cell/C` (c) Original trajectory of `Cell/X` (d) Original trajectory of `Cell/Endosome/C`

(e) Smoothed trajectory of `Cell/R` (f) Smoothed trajectory of `Cell/C` (g) Smoothed trajectory of `Cell/X` (h) Smoothed trajectory of `Cell/Endosome/C`

Figure 5.13.: Simulation results of the recycling model (ME2). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b), (c) and (d) are the original trajectories from simulation experiments; (e), (f), (g) and (h) are the smoothed trajectories using the LOESS method on the original ones, respectively. [From [186]]

5.4. Summary

In this chapter, first of all, model reuse for extension was discussed (Section 5.1). Although less addressed than model composition in the research area of model reuse, in practice model extension is a common means for developing models. By taking a previously validated model and adding new parts to capture the interesting aspects of the modeled system, the complexity in the modeling task can be reduced and the successive model extension facilitates the modeling of complex systems and the development of complex models. Based on a brief discussion of existing work in supporting model extension and a life cycle of developing models by extension, it was shown that this work aims to support the validation of the extended model, based on reusing simulation experiments.

Section 5.2 discussed how the proposed approach is used to support model extension. Applying the proposed approach for model extension is similar to the case of model composition, except that the procedure for experiment adaptation might be more straightforward in model extension, due to the fact that only one model is reused. For demonstration, a case study of modeling the receptor ligand pathway was presented, where three models were developed by successive extension. As shown in the case study, by employing the proposed approach and its prototypical implementation TAECS during the model extension, important feedback about the

validity of the extended model is provided to the user and models are developed systematically. As a result, suitable assistance can be provided for the intricate process of successively extending and validating models.

Chapter 6

Conclusions and Outlook

6.1. Summary

Model reuse has been a longstanding research topic in the modeling and simulation area, as it presents the appealing intuition of reducing the required time and cost in model development. However, in practice developing new models by reusing existing ones turns out to be a challenging endeavor, which requires suitable assistance. The modeling process is typically intertwined with the execution of simulation experiments. Thus, this thesis investigates how to employ simulation experiments to support model development by reuse.

As a starting point, Chapter 2 takes a closer look at simulation experiments and identifies different types of simulation experiments that could occur in a simulation study. Whereas simulation experiments are often performed with a model that is already satisfactorily built to fulfill the purpose of the simulation study, this work addresses those simulation experiments that are conducted for the creation of a satisfactory model, including experiments for model calibration and validation.

While numerous approaches have been developed to support the experimentation process, some effort, as well as this work, promotes the idea of treating experiments as first class objects, where simulation experiments are managed in a way so that they can be shared, reproduced, and reused. To reproduce and reuse simulation experiments, a prerequisite is an explicit, unambiguous experiment specification that is separated from model description. Based on a discussion of various approaches for specifying simulation experiments, the embedded domain specific language SESSL proposed by Ewald et al. [67] has been demonstrated to be a promising candidate. Initially designed to facilitate the set up and execution of simulation experiments, SESSL specifies experiments in a declarative manner, supports experiment reuse across different simulation systems, and, more importantly, allows an easy extension.

Thereby, Chapter 2 establishes the basis of this work, with a conclusion that simulation experiments can play an important role during model development, and an unambiguous, explicit specification of simulation experiments in SESSL facilitates their reuse.

Chapter 3 firstly discusses what kind of support is needed to facilitate model reuse and how simulation experiments can be employed to provide such support. On the one hand, efficient model reuse requires information about model context and the corresponding desired model behavior. On the other hand, simulation experiments conducted for model validation are capable of representing the model context through experiment conditions and reflecting the desired model behavior through experiment results. Therefore, simulation experiments for model validation present themselves as a useful means for model reuse, and thus become the focus of this work.

The second part of Chapter 3 presents the central concept of exploiting simulation experiments to support developing models via reuse. The essential idea is to specify simulation experiments that have been conducted for model validation and annotate models with their experiment specifications. When models are reused to build new models, those experiment specifications are also reused to perform experiments with the new model, to observe the changes in model behavior resulting from the model reuse. Hence, information is provided on whether some key model behavioral properties of individual models that should be preserved are disrupted by the model reuse, and those that should be violated still hold in the new model. This approach shares some similarities with regression testing in software development, where previously created test cases are reused when some changes have been made in the software, to make sure those changes do no affect existing functionalities of the software.

Based on the discussion in Chapter 2, an explicit experiment specification in SESSL is employed to facilitate reusing simulation experiments of individual models. In particular, to convey the information of model validity, the expected model behavior should be described explicitly, in addition to the experiment condition to generate this behavior. Therefore, four aspects are distinguished in specifying simulation experiments: model configuration, simulation configuration, data processing method, and model behavioral property. Whereas the original version of SESSL supports the specification of the first two aspects, i.e., model configuration and simulation configuration, this work extends it with a new trait to facilitate the specification of the last two aspects, i.e., data processing method and model behavioral property.

However, experiment specifications of reused models cannot be directly executed on the new model, and certain adaptation is necessary. First of all, the model

location in the experiment specification needs to be updated, from the reused model to the new model. In addition, adaptation in model configuration and property specification may be needed when model parameters are renamed or reconfigured during the creation of the new model, and those changes are defined by the user in the adaptation information. Based on experiment specifications of individual models in SESSL and specified adaptation information, a mechanism is developed to automatically perform experiments with the new model, by adapting and executing the reused experiment specifications. To facilitate an automated result analysis, the idea from simulation-based model checking techniques is adopted, which evaluates simulation results against the specified desired property with checking algorithms. Thus, different languages such as temporal logics can be employed to describe expected behavioral properties and corresponding checking algorithms are used for result analysis. Besides, due to the extensibility of SESSL, other methods for property specification and checking can be easily integrated, such as self-defined predicates or new property specification language, e.g., [260]. Also, modelers can additionally specify the expectation of property checking on the new model so that the checking results are compared with the expectation, to show whether the reuse of individual models for building the new model works as intended.

The conceptual architecture of the proposed approach is presented (Section 3.5.5), which consists of three layers: the user interface, experimentation, and result analysis. The uppermost layer allows users to specify important ingredients: the model to experiment, the experiment specification to reuse, adaptation information, and result expectation. Taking those ingredients from the first layer, the experimentation layer performs simulation experiments, and pre-processes simulation results with data processing method if specified. The bottom layer checks simulation trajectories against specified properties and returns results.

As a prototypical implementation, the tool TAECS has been developed based on SESSL, which fulfills the adaptation, execution, and checking of simulation experiments (see Section 3.5.5). TAECS automatically generates new SESSL experiment specifications by adapting reused ones, and returns them to the user. This automatic reuse, adaptation, and documentation of experiment specifications facilitate the successive reuse of models.

To apply the developed approach in supporting model development via reuse, two types of model reuse are individually considered. Chapter 4 investigates the use of the approach in model composition. A life cycle is presented to structure the intricate process of developing models by composition into several phases. Based on this life cycle, the specific phase that this work aims to support is illuminated, i.e., semantic

validation of the composed model. A composed model contains all parameters from both model components and typically has a higher parameter dimension. Thus, adapting simulation experiments for the composed model needs to consider the possible incompatibility in parameter configuration. Two solutions are described to deal with this problem. One completely relies on user specification, which includes providing a complete model configuration in the definition of the composed model and specifying the adaptation information for resolving possible conflicts. Another solution aims to provide some assistance by automatically generating assignment ranges for model parameters, based on their configuration in experiment specifications of model components.

Two case studies are presented to demonstrate the effectiveness of the proposed approach in supporting model composition: the case study of Lotka-Volterra models (Section 4.4) and the case study of Wnt/β-catenin models (Section 4.5). Based on a simple variant of the Lotka-Volterra model, the first case study illustrates how the proposed approach is used during the development of composed models. In the second case study, a realistic simulation study of modeling the Wnt/β-catenin [87, 89] is replayed, where a complex model is developed by successive composition of three model components. As illustrated in the two case studies, by providing important insights into the behavior of the composed model, the developed approach and its prototypical implementation TAECS are able to support the development of valid composed models and contribute to the process of successive model composition.

Chapter 5 is concerned with another type of model reuse, i.e., model extension. Model reuse for extension receives less attention than model composition in the research area of model reuse, and therefore this work firstly addresses this type of reuse. Through a discussion of developing models via extending existing ones, the benefits of this type of model reuse are shown, which include reducing the complexity of the modeling task, and facilitating the development of large, complex models. Subsequently, the proposed approach is applied to the model development via extension, aiming to support the validation of the extended model. For demonstration, a case study of modeling the receptor ligand pathway is presented, where three models are developed via extending one after another. As shown in the case study, by automatically reusing simulation experiments performed with the original model, the proposed approach along with TAECS can help to systematically develop extended models and provide valuable support in the process of successive model extension.

From the perspective of methodology, applying the proposed approach in model composition and in model extension do not make much difference, except that in model composition the adaptation of experiment specifications might need more

effort (see Section 4.3.2). In model composition, two models are reused to build the composed model, and their composition may result in emergent behavior, which is not easily predictable by individually analyzing the behavior of each reused model component. The proposed approach focuses on behavioral properties that are exhibited by individual reused models, and inspects the impact of model reuse on those properties. Therefore, further work toward identifying and validating emergent properties is needed to facilitate the development of valid composed models. When models are built by extending existing ones, only one model is reused. By adding new parts into the reused model, model extension may lead to new behavior in the extended model, in addition to possible changes in the existing behavior exhibited by the original model. Whereas the developed approach assists modelers in ensuring that the key behavior of the original model is not disrupted by the extension and the obsolete behavior is not preserved, additional effort is required to validate the new behavior, e.g., by performing new simulation experiments. Independently of whether the model reuse leads to emergent behavior or new behavior, when the existing behavioral properties are influenced by the composition or extension, either as expected or not, the proposed approach can uncover those changes and thus provide modelers with important feedback.

Based on two types of model reuse and altogether three case studies, the effectiveness of the proposed approach along with the developed tool TAECS has been demonstrated. As SESSL is independent of concrete modeling approaches and simulation systems, the proposed approach inherits this merit. For demonstration, another example of Lotka-Volterra models is presented in Appendix A, where SBML, one of the widely used modeling formalisms for systems biology, is employed to describe models and its listed simulation tool SBMLSim is used to perform simulation experiments. Due to the extensibility of SESSL, various methods for property specification and property checking, as well as data processing methods, can be integrated. So far, $\text{MITL}_{[a,b]}$ (in the case study of Wnt/β-catenin pathway, Section 4.5, and the case study of receptor ligand pathway, Section 5.3), LTL and custom predicate definition (in the case study of Lotka-Volterra models, Section 4.4) have been used for describing individual simulation trajectories. To deal with the stochasticity arising from experiments with stochastic models, CSL has been employed to express probabilistic properties regarding replications and statistical model checking methods were used for probability estimation (in the case study of Lotka-Volterra models and the case study of receptor ligand pathway). Moreover, averaging over multiple replications was used, when model behavioral properties were derived based on

aggregated observations and high variance exists among different simulation runs (in the case study of Wnt/β-catenin pathway).

To summarize, this thesis presents an approach of reusing simulation experiments of individual models to automatically perform experiments with the model that is developed by composing or extending existing ones. As a result, modelers can gain crucial insights into whether individual models are reused as intended and whether the newly built model displays valid behavior. With the proposed approach and its prototypical implementation TAECS, a fast and valuable feedback is provided to modelers during the model reuse, and new models are developed in a systematic way. Hence, suitable assistance can be provided to accelerate the modeling process and improve the quality of model development.

6.2. Limitations and Future Work

One limitation of this work is that all the information required for adaptation of simulation experiments is specified by users. Although some assistance can be provided in the case of model composition by generating parameter assignment ranges (see Section 4.3.2), users still need to define a complete model configuration in the model file and specify the necessary information for adaptation. Thus, further effort could be an automated extraction of adaptation information by monitoring and analyzing the differences between the reused models and the resulting model. Relevant work includes model version control (e.g., [257]) for detecting changes in models, and model transformation (e.g., [161]) for mapping differences in models to differences in experiment specifications.

One prerequisite of this work is an explicit specification of model behavioral properties that are essential to the model validity, and an automatic result analysis against the specified properties. When individual models are reused to build new ones, regarding the same property of interest the behavior displayed by the newly built model might be different from the reused model. However, it highly depends on users' interpretation to answer the question of whether the dissimilarity is "slight" enough so that the behavior of both models should be perceived as equally valid, or it should be considered as a violation of this behavioral property by the new model. For example, in the reused model, one behavioral property is that under a certain experiment condition the model variable `A` reaches steady state with the value being 10, whereas in the new model, under the same experiment condition, the model variable `A` also reaches steady state but with the value being 11. Without additional information from users, it is difficult to determine whether 10 and 11

are sufficiently close so that the new model still satisfies the behavioral property of the reused model, or this property is violated by the new model. Thus, to more accurately assess model behavior, the specification of model behavioral properties needs to take this uncertainty into account, for which, however, there seems to be no easy solution and users' intervention appears necessary.

Simulation experiments that are performed for model validation are the focus of this work. As stated in Section 3.1.2, various techniques exist to validate models based on performing simulation experiments, such as face validation, each checking the model behavior against certain requirements. Currently, in this work, applicable simulation experiments are limited to those where requirements on model behavior can be reflected by simulation trajectories and explicitly expressed in properties. Thus, another direction of future work could be to support other types of validation experiments, e.g., sensitivity analysis, for which methods to express corresponding requirements on model behavior are needed.

The ultimate goal of this work is to provide support for developing models based on model reuse, with a focus on validation of the resulting model. However, the modeling process is highly complex and efforts on solely one specific aspect of this process can only provide limited support. Therefore, to better facilitate the development of models, this work should be combined with other efforts toward this goal.

One possible direction is to integrate this approach into management of the modeling process. An artifact-based workflow approach is proposed by Rybacki et al. [208], to structure the modeling process and further manage it. Three artifacts are presented in [208], and the artifact of formal model is of relevance, as this work focuses on the development of formal models. To integrate the developed approach, the artifact Formal Model can be adapted as depicted in Figure 6.1. Two stages are defined in this artifact, i.e., `Creating Formal Model` and `V&V Formal Model`, which represent the creation of the formal model and its verification and validation (V&V), respectively. This work can be viewed as a sub-stage in the stage `V&V Formal Model`, namely the `Validating via Reusing Experiments` sub-stage. This sub-stage can be activated when the formal model is created based on reusing previously validated models and those reused models are annotated with specifications of their validation experiments. The integration of the proposed approach with artifact-based workflow is ongoing work, with the first results presented in [205].

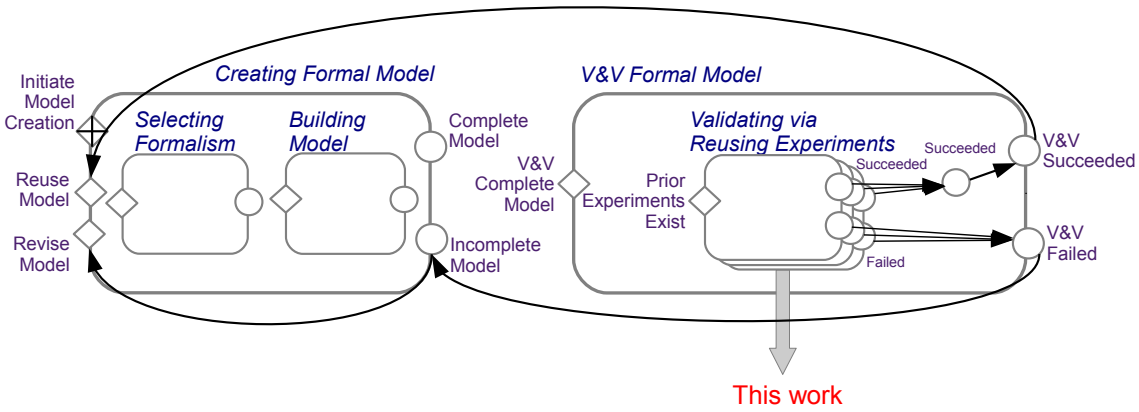


Figure 6.1.: The adapted artifact Formal Model in the artifact-based workflow approach proposed by Rybacki et al. [208]. This work provides assistance in validating the formal model based on prior simulation experiments, and thus presents as a sub-stage of the stage V&V Formal Model, i.e., *Validating via Reusing Simulation Experiments*. This sub-stage can be activated when the model is created by reusing previously validated models and simulation experiments for validating the reused models are available.

Appendix A

An Example of Composing Lotka-Volterra models Based on the SBML formalism

To demonstrate that the presented approach is independent of concrete modeling and simulation methods, the Lotka-Volterra models described in Case Study I (Section 4.4) are used, however, using the modeling formalism SBML [103], instead of ML-Rules.

In this example, for simplicity, parameters are configured as single constants, rather than assignment ranges in Case Study I. The initial predator and prey population size as 10 and 100, respectively, and the rest model parameters are configured as follows:

$$a : 0.014, b : 0.6, d : 0.7, k : 0.002$$

For a model with rabbits as the prey and foxes as the predator (M1), the reaction rules are depicted in Figure 4.3. The formalism SBML (Systems Biology Markup Language) [103] is used to represent the model (model files can be found in the supplemental package). The SBMLSim, one of the listed tools for SBML models [245], is selected for model execution.

Performing simulation experiments with SBML models based on SBMLSim has been supported by SESSL through the binding `sessl.sbmlsim`. To run simulation experiments, the provided simulator `DormandPrice54` from SBMLSim is used, with the step size being 0.01. Simulation stops after 10 simulation time units and during the simulation the number of the prey and predator are observed at each 0.1 time unit. Models represented in SBML are deterministic, and thus only one replication is necessary. Simulation results are shown in Figure A.1.

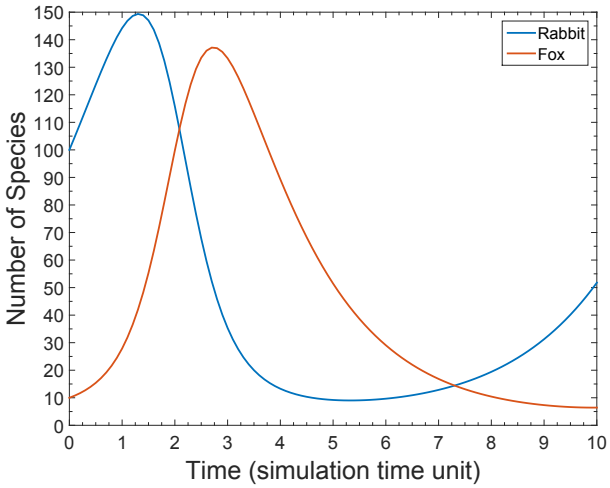


Figure A.1.: Simulation results of the basic Lotka-Volterra model described in SBML, where rabbits are the prey and foxes are the predator.

According to the simulation result, one model behavioral property was derived that both the prey and the predator do not die out, shown as follows:

$$G(\#Rabbit > 0 \wedge \#Fox > 0)$$

Based on the simulation set-up and the property, the experiment is specified in SESSL, as shown in Figure A.2.

```
import sessl._
import sessl.sbmlsim._
val exp = new Experiment with Observation with Hypotheses {
  model = "file-mlrj:/./LotkaVolteraM1.mlrj"
  simulator = DormandPrince54(0.01)
  stopTime = 10.0
  observe("Rabbit", "Fox")
  observeAt(range(0.0, 0.1, 10.0))
  assume(
   G(0.0, 10.0, variable("Rabbit") > 0 and variable("Fox") > 0)
   )
}
```

Figure A.2.: SESSL experiment specification for the basic Lotka-Volterra model described in SBML.

Similarly, another basic Lotka-Volterra model M2 is needed, where rabbits are the prey and wolves are the predator. Using the same model configuration and experiment set-up, model M2 also satisfies the behavioral property that both the prey (rabbits) and the predator (wolves) survive. Thereby, the basic Lotka-Volterra model M2 is annotated with an experiment specification, which is similar to Figure A.2, except that the species fox is replaced by the wolf.

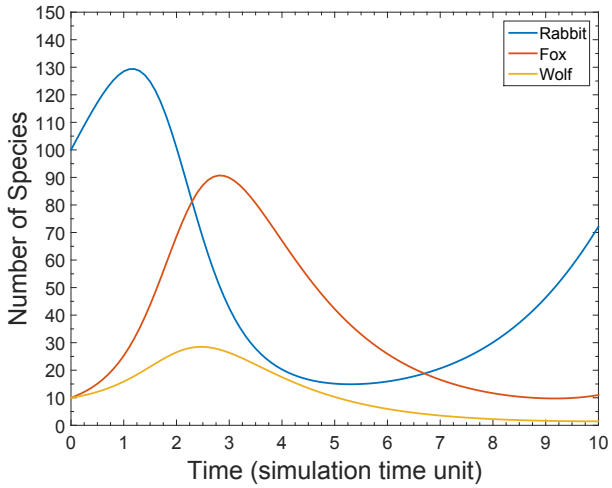


Figure A.3.: Simulation results of the composed Lotka-Volterra model described in SBML.

Based on the two basic Lotka-Volterra models, one with rabbits and foxes (M1) and another with rabbits and wolves (M2), a composed model M12 can be developed, where all three species exist and both foxes and wolves hunt on rabbits. The composed model is created by integrating the two model components together and the reaction rules are described in Figure 4.10(a). The parameter configuration remains the same as in model components and no new parameter is introduced.

Experiment specifications of the two model components are reused to perform experiments on the composed model. As no parameters are renamed or reconfigured during the composition, in the reused experiment specifications only the model location is updated from the model component to the composed model. Based on the adapted experiment specifications, experiments are automatically executed with the composed model and simulation outputs are checked against the specified behavioral property. Simulation trajectories are shown in Figure A.3. Test results show that the property of both model components hold for the composed model.

List of Figures

List of Tables

Bibliography

[1] R. R. Adams. SED-ED, a workflow editor for computational biology experiments written in SED-ML. *Bioinformatics*, 28(8):1180–1181, 2012.

[2] B. Alpern and F. B. Schneider. Verifying temporal properties without temporal logic. *ACM Transactions on Programming Languages and Systems*, 11(1):147–167, Jan. 1989.

[3] R. Alur, T. Feder, and T. A. Henzinger. The Benefits of Relaxing Punctuality. *J. ACM*, 43(1):116–146, 1996.

[4] J. G. Arnold, D. N. Moriasi, P. W. Gassman, et al. SWAT: Model use, calibration, and validation. *Transactions of the ASABE*, 55(4):1491–1508, 2012.

[5] F. Bacchus, J. Tenenberg, and J. A. Koomen. A non-reified temporal logic. *Artificial Intelligence*, 52:87–108, 1991.

[6] O. Balci. Guidelines for successful simluation studies (tutorial session). In *Proceedings of the 22nd winter simulation conference*, pages 25–32. IEEE Press, 1990.

[7] O. Balci. Verification validation and accreditation of simulation models. In *Proceedings of the 29th winter simulation conference*, pages 135–141. IEEE Computer Society, 1997.

[8] O. Balci. Verification, validation, and testing. In J. Banks, editor, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, pages 335–393. John Wiley & Sons, 1998.

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

[10] O. Balci, J. D. Arthur, and R. E. Nance. Accomplishing reuse with a simulation conceptual model. In *Proceedings of the 2008 winter simulation conference*, pages 959–965. IEEE, 2008.

[11] P. Ballarini, J. Mäkelä, and A. S. Ribeiro. Expressive statistical model checking of genetic networks with delayed stochastic dynamics. In D. Gilbert and M. Heiner, editors, *Computational Methods in Systems Biology*, Lecture Notes in Computer Science, pages 29–48. Springer Berlin Heidelberg, 2012.

[12] J. Banks. *Discrete-event System Simulation.* Discrete-event System Simulation. Prentice Hall, 2001.

[13] A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

[14] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer Berlin Heidelberg, 2004.

[15] R. G. Bartholet, D. C. Brogan, P. F. Reynolds Jr, and J. C. Carnahan. In search of the philosopher's stone: Simulation composability versus component-based software design. In *Proceedings of the Fall Simulation Interoperability Workshop*, 2004.

[16] R. R. Barton. Designing simulation experiments. In *Proceedings of the 2013 winter simulation conference*, pages 342–353. IEEE Press, 2013.

[17] G. Batt, J. T. Bradley, R. Ewald, F. Fages, et al. 06161 working groups' report: The challenge of combining simulation and verification. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[18] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 260–272. Springer, 2006.

[19] H. Benali and N. B. Ben Saoud. Towards a component-based framework for interoperability and composability in modeling and simulation. *Simulation*, 87 (1-2):133–148, 2011.

[20] F. T. Bergmann, R. Adams, S. Moodie, J. Cooper, et al. COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project. *BMC bioinformatics*, 15(1):1, 2014.

[21] J. Bézivin, S. Bouzitouna, M. D. Del Fabro, et al. A canonical scheme for model composition. In *Model Driven Architecture–Foundations and Applications*, pages 346–360. Springer, 2006.

[22] A. Bouajjani and Y. Lakhnech. Temporal logic + timed automata: Expressiveness and decidability. In I. Lee and S. Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 531–545. Springer Berlin Heidelberg, 1995.

[23] A. Bouajjani and Y. Lakhnech. Logics vs. automata: The hybrid case. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 531–542. Springer Berlin Heidelberg, 1996.

[24] G. E. Box. Robustness in the strategy of scientific model building. *Robustness in statistics*, 1:201–236, 1979.

[25] P. Bratley, B. Fox, and L. Schrage. *A Guide to Simulation*. Springer New York, 2012.

[26] L. Brim, M. Ceska, S. Drazan, and D. Safránek. On robustness analysis of stochastic biochemical systems by probabilistic model checking. *CoRR*, abs/1310.4734, 2013.

[27] L. Brim, M. Češka, and D. Šafránek. Model checking of biological systems. In *Formal Methods for Dynamical Systems*, pages 63–112. Springer, 2013.

[28] L. Calzone, F. Fages, and S. Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14): 1805–1807, 2006.

[29] M. Carrillo, P. A. Góngora, and D. A. Rosenblueth. An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. *Frontiers in plant science*, 3:155, 2012.

[30] F. Cellier. *Continuous System Modeling*. Continuous System Modeling. Springer, New York, 1991.

[31] C. Cercignani and M. Lampis. Kinetic models for gas-surface interactions. *Transport Theory and Statistical Physics*, 1(2):101–114, 1971.

[32] R. Chreyh and G. Wainer. Cd++ repository: an internet based searchable database of devs models and their experimental frames. In *Proceedings of the 2009 Spring Simulation Multiconference*, page 159. Society for Computer Simulation International, 2009.

[33] L. Chwif, J. Banks, J. P. de Moura Filho, and B. Santini. A framework for specifying a discrete-event simulation conceptual model. *Journal of Simulation*, 7(1):50–60, 2013.

[34] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems*, pages 147–188. Sp, Berlin Heidelberg, 2004.

[35] E. Clarke, A. Donzé, and A. Legay. On simulation-based probabilistic model checking of mixed-analog circuits. *Formal Methods in System Design*, 36(2):97–113, 2010.

[36] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, Mass. [u.a.], 1999.

[37] E. M. Clarke, J. R. Faeder, C. Langmead, et al. Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway. In *Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 231–250. Sp, Berlin Heidelberg, 2008.

[38] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, 1979.

[39] H. Clevers and R. Nusse. Wnt/β-catenin signaling and disease. *Cell*, 149(6):1192–1205, 2012.

[40] Commons Math Developers. Apache commons math, release 3.6. Available from `https://commons.apache.org/proper/commons-math/download_math.cgi`, 2016.

[41] J. Cooper, G. R. Mirams, and S. A. Niederer. High-throughput functional curation of cellular electrophysiology models. *Progress in biophysics and molecular biology*, 107(1):11–20, 2011.

[42] J. Cooper, J. O. Vik, and D. Waltemath. A call for virtual experiments: accelerating the scientific process. *Progress in biophysics and molecular biology*, 117(1):99–106, 2015.

[43] J. Cooper, M. Scharm, and G. R. Mirams. The cardiac electrophysiology web lab. *Biophysical journal*, 110(2):292–300, 2016.

[44] S. A. Coskun, A. E. Cicek, N. Lai, et al. An online model composition tool for system biology models. *BMC systems biology*, 7(1):88, 2013.

[45] M. Courtot, N. Juty, Knüpfer, et al. Controlled vocabularies and semantics in systems biology. *Molecular systems biology*, 7(1):543, 2011.

[46] J. O. Dada, I. Spasić, N. W. Paton, and P. Mendes. Sbrml: a markup language for associating systems biology data with models. *Bioinformatics*, 26(7):932–938, 2010.

[47] D. Dahmen, H. E. Plesser, and S. Kunkel. Replicability and reproducibility of neural network simulations. In *Berstein Conference*, 2015.

[48] O. Dalle. *OSA: An open component-based architecture for discrete-event simulation.* PhD thesis, INRIA, 2005.

[49] O. Dalle. On reproducibility and traceability of simulations. In *Proceedings of the 2012 winter simulation conference*, pages 1–12. IEEE, 2012.

[50] T. Daum and R. G. Sargent. Experimental frames in a modern modeling and simulation system. *IIE Transactions*, 33(3):181–192, 2001.

[51] P. K. Davis and R. H. Anderson. Improving the composability of DoD models and simulations. *JDMS*, 1(1):5–17, Apr. 2004.

[52] P. K. Davis and A. Tolk. Observations on New Developments in Composability and Multi-resolution Modeling. In *Proceedings of the 39th winter simulation conference*, WSC '07, pages 859–870, Piscataway, NJ, USA, 2007. IEEE Press.

[53] L. De Alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, M. Broy, J. Gruenbauer, D. Harel, and C.A.R. Hoare, 2005.

[54] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of the 23rd Int. Joint Conference on Artificial Intelligence*, pages 854–860. AAAI Press, 2013.

[55] J.-L. de Kok, G. Engelen, and J. Maes. Reusability of model components for environmental simulation–case studies for integrated coastal zone management. *Environmental Modelling & Software*, 68:42–54, 2015.

[56] H. de Swaan Arons and C. A. Boer. Storage and retrieval of discrete-event simulation models. *Simulation Practice and Theory*, 8(8):555–576, 2001.

[57] M. del Milagro Gutierrez and H. P. Leone. Composability model in a distributed simulation environment for supply chain. *Iberoamerican Journal of Industrial Engineering*, 5(10):55–69, 2014.

[58] B. Delinchant, F. Wurtz, D. Magot, and L. Gerbaud. A component-based framework for the composition of simulation software modeling electrical systems. *Simulation*, 80(7-8):347–356, 2004.

[59] J. M. Donohue. Experimental designs for simulation. In *Proceedings of the 26th winter simulation conference*, pages 200–206. Society for Computer Simulation International, 1994.

[60] D.R. Miller. Sensitivity analysis and validation of simulation models. *Journal of Theoretical Biology*, 48(2):345 – 360, 1974.

[61] M. Droz and A. Pękalski. Different strategies of evolution in a predator-prey system. *Physica A: Statistical Mechanics and its Applications*, 298:545–552, 2001.

[62] P. Du, W. A. Kibbe, and S. M. Lin. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065, 2006.

[63] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Periodicity detection in time series databases. *IEEE Transactions on Knowledge and Data Engineering*, 17:1–13, 2005.

[64] H. Elmqvist, S. E. Mattsson, and M. Otter. Object-oriented and hybrid modeling in modelica. *Journal Européen des systèmes automatisés*, 35(1):1–10, 2001.

[65] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, 2010.

[66] R. Ewald and A. M. Uhrmacher. Automating the runtime performance evaluation of simulation algorithms. In *Proceedings of the 2009 winter simulation conference*, pages 1079–1091. IEEE, 2009.

[67] R. Ewald and A. M. Uhrmacher. SESSL: A domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation*, 24(2): 11:1–11:25, Feb. 2014.

[68] R. Ewald, J. Himmelspach, and A. M. Uhrmacher. An Algorithm Selection Approach for Simulation Systems. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS'08)*, pages 91–98, 2008.

[69] R. Ewald, J. Himmelspach, M. Jeschke, S. Leye, and A. M. Uhrmacher. Flexible experimentation in the modeling and simulation framework JAMES II–Implications for computational systems biology. *Briefings in bioinformatics*, 11(3):290–300, 2010.

[70] F. Fages and A. Rizk. On the analysis of numerical data time series in temporal logic. In *Computational Methods in Systems Biology*, 2007.

[71] F. Fages and A. Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theoretical Computer Science*, 408(1):55–65, 2008.

[72] H. Fan and A. E. Mark. Refinement of homology-based protein structures by molecular dynamics simulation techniques. *Protein Science*, 13(1):211–220, 2004.

[73] J. Fisher, D. Harel, and T. A. Henzinger. Biology as Reactivity. *CACM*, 54(10): 72–82, 2011.

[74] A. Fletcher, D. Halsall, S. Huxham, and D. Worthington. The dh accident and emergency department model: a national generic model used locally. *Journal of the Operational Research Society*, 58(12):1554–1562, 2007.

[75] Y. Funato, T. Michiue, M. Asashima, and H. Miki. The thioredoxin-related redox-regulating protein nucleoredoxin inhibits Wnt–β-catenin signalling through dishevelled. *Nature cell biology*, 8(5):501–508, 2006.

[76] J. H. Gennari, M. L. Neal, M. Galdzicki, and D. L. Cook. Multiple ontologies in action: composite annotations for biosimulation models. *Journal of biomedical informatics*, 44(1):146–154, 2011.

[77] D. Ghosh. *DSLs in Action*. Manning Publications Co., Greenwich, CT, USA, 2011.

[78] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[79] L. Ginzburg and H. Akçakaya. Consequences of ratio-dependent predation for steady-state properties of ecosystems. *Ecology*, 73(5):1536–1543, 1992.

[80] E. Glinsky and G. Wainer. Devstone: a benchmarking technique for studying performance of devs modeling and simulation environments. In *Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 265–272. IEEE, 2005.

[81] W. Gong, Y. M. Tien, C. H. Juang, J. R. Martin, and J. Zhang. Calibration of empirical models considering model fidelity and model robustness—focusing on predictions of liquefaction-induced settlements. *Engineering Geology*, 2015.

[82] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback. The ODD protocol: A review and first update. *Ecological Modelling*, 221(23): 2760–2768, 2010.

[83] V. Grimm, P. Polhill, and J. Touza. Documenting social simulation models: The odd protocol as a standard. In *Simulating Social Complexity - A handbook*, pages 117–133. Springer, 2013.

[84] M. Grohe. Generalized Model-Checking Problems for First-Order Logic. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '01, pages 12–26, London, UK, UK, 2001. Springer-Verlag.

[85] T. Gruber. Ontology. `http://tomgruber.org/writing/ontology-definition-2007.htm`. Accessed: 2016-10-17.

[86] P. L. Gustavson, J. P. Hancock, and M. McAuliffe. Base Object Models (BOMs): Reusable Component Objects for Federation Development. In *Simulation Interoperability Workshop*, 1998.

[87] F. Haack. *Exploring the spatio-temporal dynamics of lipid rafts and their role in signal transduction: a modeling and simulation approach*. PhD thesis, University of Rostock, 2016.

[88] F. Haack, K. Burrage, R. Redmer, and A. M. Uhrmacher. Studying the role of lipid rafts on protein receptor bindings with cellular automata. *IEEE/ACM transactions on computational biology and bioinformatics*, 10(3):760–770, 2013.

[89] F. Haack, H. Lemcke, R. Ewald, et al. Spatio-temporal model of endogenous ros and raft-dependent wnt/beta-catenin signaling driving cell fate commitment in human neural progenitor cells. *PLoS Comput Biol*, 11(3):1–28, 03 2015.

[90] A. Hallagan, B. Ward, and L. F. Perrone. An experiment automation framework for ns-3. In *Proceedings of the 3rd Int'l ICST Conference on Simulation Tools and Techniques*. ICST, 2010.

[91] R. N. Hannoush. Kinetics of wnt-driven β-catenin stabilization revealed by quantitative and temporal imaging. *PloS ONE*, 3(10):e3498, 2008.

[92] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[93] T. Helms, M. Luboschik, H. Schumann, and A. Uhrmacher. An approximate execution of rule-based multi-level models. In A. Gupta and T. Henzinger, editors, *Computational Methods in Systems Biology*, volume 8130 of *Lecture Notes in Computer Science*, pages 19–32. Springer, Berlin Heidelberg, 2013.

[94] T. Helms, C. Maus, F. Haack, and A. M. Uhrmacher. Multi-level modeling and simulation of cell biological systems with ml-rules: A tutorial. In *Proceedings of the 2014 winter simulation conference*, WSC '14, pages 177–191, Piscataway, NJ, USA, 2014. IEEE Press.

[95] T. Helms, R. Ewald, S. Rybacki, and A. M. Uhrmacher. Automatic runtime adaptation for component-based simulation algorithms. *Acm Transactions on Modeling & Computer Simulation*, 26(1):1–24, 2015.

[96] R. Henkel, O. Wolkenhauer, and D. Waltemath. Combining computational models, semantic annotations and simulation experiments in a graph database. *Database*, 2015:bau130, 2015.

[97] J. Hillston. A tool to enhance model exploitation. *Performance evaluation*, 22(1): 59–74, 1995.

[98] J. Himmelspach. *Konzeption, Realisierung und Verwendung eines allgemeinen Modellierungs-, Simulations-und Experimentiersystems: Entwicklung und Evaluation effizienter Simulationsalgorithmen*. Sierke, 2007.

[99] J. Himmelspach and A. M. Uhrmacher. Plug'N Simulate. In *Proceedings of the 40th Annual Simulation Symposium*, ANSS '07, pages 137–143, Washington, DC, USA, 2007. IEEE Computer Society.

[100] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'06, pages 441–444, Berlin, Heidelberg, 2006. Springer-Verlag.

[101] K. Hoad, S. Robinson, and R. Davies. AutoSimOA: A framework for automated analysis of simulation output. *Journal of Simulation*, 5(1):9–24, 2011.

[102] M. Hofmann. On the complexity of parameter calibration in simulation models. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 2(4):217–226, 2005.

[103] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[104] P. Kacsuk. Enabling distributed simulations using big data and clouds. In *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*, pages 125–126, New York, NY, USA, 2015. ACM.

[105] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, et al. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150(2):389–401, 2012.

[106] S. Kasputis and H. C. Ng. Model composability: formulating a research thrust: composable simulations. In *Proceedings of the 32nd winter simulation conference*, pages 1577–1584. Society for Computer Simulation International, 2000.

[107] J.-P. Katoen and I. S. Zapreev. Simulation-based ctmc model checking: An empirical evaluation. In *Sixth International Conference on the Quantitative Evaluation of Systems. QEST'09.*, pages 31–40. IEEE, 2009.

[108] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance evaluation*, 68(2): 90–104, 2011.

[109] W. Kelton, R. Sadowski, and N. Swets. *Simulation with Arena*. McGraw-Hill Education, 2015.

[110] W. D. Kelton. Design of experiments: experimental design for simulation. In *Proceedings of the 32nd winter simulation conference*, pages 32–38. Society for Computer Simulation International, 2000.

[111] W. D. Kelton and R. R. Barton. Experimental design for simulation: Experimental design for simulation. In *Proceedings of the 35th winter simulation conference*, WSC '03, pages 59–65. Winter Simulation Conference, 2003.

[112] J. Kleijen. Validation of models: statistical techniques and data availability. In *Proceedings of the 1999 winter simulation conference*, volume 1, pages 647–654. IEEE, 1999.

[113] J. Kleijnen. *Design and Analysis of Simulation Experiments*. International Series in Operations Research & Management Science. Springer US, 2007.

[114] J. P. Kleijnen. Verification and validation of simulation models. *European journal of operational research*, 82(1):145–162, 1995.

[115] J. P. Kleijnen. Experimental design for sensitivity analysis, optimization, and validation of simulation models. In J. Banks, editor, *Handbook of Simulation:*

Principles, Methodology, Advances, Applications, and Practice, pages 173–223. John Wiley & Sons, 1998.

[116] J. P. Kleijnen, S. M. Sanchez, T. W. Lucas, and T. M. Cioppa. State-of-the-art review: a user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, 17(3):263–289, 2005.

[117] D. Köhn, C. Maus, R. Henkel, and M. Kolbe. Towards enhanced retrieval of biological models through annotation-based ranking. In *International Workshop on Data Integration in the Life Sciences*, pages 204–219. Springer, 2009.

[118] L. F. Konikow and J. D. Bredehoeft. Ground-water models cannot be validated. *Advances in water resources*, 15(1):75–83, 1992.

[119] G. Korn and J. Wait. *Digital continuous-system simulation*. Prentice-Hall, 1978.

[120] F. Krause, J. Uhlendorf, T. Lubitz, et al. Annotation and merging of SBML models with semanticSBML. *Bioinformatics*, 26(3):421–422, 2010.

[121] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: the incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50–61, 2005.

[122] A. D. Lattner, H. Pitsch, I. J. Timm, et al. Assistsim–towards automation of simulation studies in logistics. *Simulation Notes Europe*, 21(3-4):119–128, 2011.

[123] D. A. Lauffenburger and J. J. Linderman. *Receptors: models for binding, trafficking, and signaling*. Oxford University Press, New York, 1996.

[124] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill international series. McGraw-Hill, 2000.

[125] A. M. Law. A tutorial on design of experiments for simulation modeling. In *Proceedings of the 2014 winter simulation conference*, WSC '14, pages 66–80, Piscataway, NJ, USA, 2014. IEEE Press.

[126] J. Ledet, S. Cam, B. K. Gorur, O. Dayibas, H. Oguztuzun, L. Yilmaz, and A. E. Smith. A hybrid transformation process for simulation modernization and reuse via model replicability and scenario reproducibility. *Transformation*, 7:8, 2015.

[127] E. Lee, A. Salic, R. Krüger, et al. The roles of apc and axin derived from experimental and theoretical analysis of the wnt pathway. *PLoS Biol*, 1(1):e10, 2003.

[128] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *Runtime Verification*, pages 122–135, 2010.

[129] H. K. Leung and L. White. Insights into regression testing [software testing]. In *Proceedings of the Conference on Software Maintenancen*, pages 60–69. IEEE, 1989.

[130] J. Lewis, C. E. Breeze, J. Charlesworth, O. J. Maclaren, and J. Cooper. Where next for the reproducibility agenda in computational biology? *BMC Systems Biology*, 10 (1):52, 2016.

[131] S. Leye. *Toward guiding simulation experiments*. PhD thesis, University of Rostock, 2013.

[132] S. Leye, J. Himmelspach, and A. M. Uhrmacher. A discussion on experimental model validation. In *11th International Conference on Computer Modelling and Simulation*, pages 161–167. IEEE, 2009.

[133] S. Leye, R. Ewald, and A. M. Uhrmacher. Composing problem solvers for simulation experimentation: A case study on steady state estimation. *PloS ONE*, 9(4):e91948, 04 2014.

[134] C. Li, M. Donizelli, N. Rodriguez, et al. Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC systems biology*, 4(1):92, 2010.

[135] W. Lindemann. Extension of a binaural cross-correlation model by contralateral inhibition. i. simulation of lateralization for stationary signals. *The Journal of the Acoustical Society of America*, 80(6):1608–1622, 1986.

[136] C. M. Lloyd, M. D. Halstead, and P. F. Nielsen. CellML: its future, present and past. *Progress in biophysics and molecular biology*, 85(2):433–450, 2004.

[137] B. Lloyd-Lewis, A. G. Fletcher, T. C. Dale, and H. M. Byrne. Toward a quantitative understanding of the Wnt/β-catenin pathway through simulation and experiment. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 5(4):391–407, 2013.

[138] R. Lord. Some extensions to the cercignani–lampis gas–surface scattering kernel. *Physics of Fluids A: Fluid Dynamics (1989-1993)*, 3(4):706–710, 1991.

[139] R. Lord. Some further extensions of the cercignani–lampis gas–surface interaction model. *Physics of Fluids (1994-present)*, 7(5):1159–1161, 1995.

[140] A. Lotka. *Elements of Physical Biology*. Williams & Wilkins Company, 1925.

[141] M. G. Lozano, F. Moradi, and R. Ayani. Sdr: A semantic based distributed repository for simulation models and resources. In *Modelling & Simulation, 2007. AMS'07. First Asia International Conference on*, pages 171–176. IEEE, 2007.

[142] M. Luboschik, S. Rybacki, F. Haack, and H.-J. Schulz. Supporting the integrated visual analysis of input parameters and simulation trajectories. *Computers & Graphics*, 39:37–47, 2014.

[143] G. Lucko, P. C. Benjamin, K. Swaminathan, and M. G. Madden. Comparison of manual and automated simulation generation approaches and their use for construction applications. In *Proceedings of the 2010 winter simulation conference*, pages 3132–3144. Winter Simulation Conference, 2010.

[144] J. Ma and B. Knight. A reified temporal logic. *The Computer Journal*, 39(9):800–807, 1996.

[145] D. Machado, R. S. Costa, M. Rocha, I. Rocha, B. Tidor, and E. C. Ferreira. A critical review on modelling formalisms and simulation tools in computational biosystems. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 1063–1070. Springer, 2009.

[146] D. Machado, R. S. Costa, M. Rocha, E. C. Ferreira, B. Tidor, and I. Rocha. Modeling formalisms in systems biology. *AMB Express*, 1(45):1–14, 2011.

[147] G. T. Mackulak, F. P. Lawrence, and T. Colvin. Effective simulation model reuse: a case study for amhs modeling. In *Proceedings of the 30th winter simulation conference*, pages 979–984. IEEE Computer Society Press, 1998.

[148] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi. Statemachine matching in bom based model composition. In *Distributed Simulation and Real Time Applications, 2009. DS-RT'09. 13th IEEE/ACM International Symposium on*, pages 136–143. IEEE, 2009.

[149] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, Heidelberg, 2004.

[150] O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*, pages 475–505. Springer, 2008.

[151] A. Maria. Introduction to modeling and simulation. In *Proceedings of the 29th winter simulation conference*, pages 7–13. IEEE Computer Society, 1997.

[152] E. D. Maria, F. Fages, and S. Soliman. On coupling models using model-checking: Effects of irinotecan injections on the mammalian cell cycle. In *Proceedings of Computational Methods in Systems Biology*, pages 142–157. Springer, 2009.

[153] C. Maus, S. Rybacki, and A. M. Uhrmacher. Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5(1):166, 2011.

[154] K. H. Mayo, M. Nunez, C. Burke, C. Starbuck, D. Lauffenburger, and C. R. Savage. Epidermal growth factor receptor binding is not a simple one-step process. *Journal of Biological Chemistry*, 264(30):17838–44, 1989.

[155] O. Mazemondet, R. Hubner, J. Frahm, et al. Quantitative and kinetic profile of Wnt/β-catenin signaling components during human neural progenitor cell differentiation. *Cellular & molecular biology letters*, 16(4):515–538, 2011.

[156] O. Mazemondet, R. Hubner, J. Frahm, et al. Quantitative and kinetic profile of Wnt/β-catenin signaling components during human neural progenitor cell differentiation. *Cellular & molecular biology letters*, 16(4):515–538, 2011.

[157] O. Mazemondet, M. John, S. Leye, A. Rolfs, and A. M. Uhrmacher. Elucidating the sources of beta-catenin dynamics in human neural progenitor cells. *PloS ONE*, 7(8): e42792–e42792, 2012.

[158] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):954–968, July 2005.

[159] J. K. Medley, A. Goldberg, and J. R. Karr. Guidelines for reproducibly building and simulating systems biology models. 2016.

[160] N. D. Mendes, F. Lang, Y.-S. Le Cornec, R. Mateescu, G. Batt, and C. Chaouiya. Composition and abstraction of logical regulatory modules: application to multicellular systems. *Bioinformatics*, 29(6):749–757, 2013.

[161] T. Mens, K. Czarnecki, and P. V. Gorp. 04101 discussion–a taxonomy of model transformations. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.

[162] M. Messer, J. H. Panchal, J. K. Allen, and F. Mistree. Model refinement decisions using the process performance indicator. *Engineering Optimization*, 43(7):741–762, 2011.

[163] M. Minsky. Models, minds, machines. In *Proceedings of IFIP Congress*, pages 45–49, 1965.

[164] G. R. Mirams, H. M. Byrne, and J. R. King. A multiple timescale analysis of a mathematical model of the wnt/β-catenin signalling pathway. *Journal of mathematical biology*, 60(1):131–160, 2010.

[165] T. Monks, S. Robinson, and K. Kotiadis. Model reuse versus model development: Effects on credibility and learning. In *Proceedings of the 2009 winter simulation conference*, pages 767–778. Winter Simulation Conference, 2009.

[166] F. Moradi, P. Nordvaller, and R. Ayani. Simulation model composition using boms. In *2006 Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 242–252. IEEE, 2006.

[167] F. Moradi, R. Ayani, and G. Tan. A rule-based approach to syntactic and semantic composition of boms. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 145–155. IEEE Computer Society, 2007.

[168] M. Mroz and G. Franks. A performance experiment system supporting fast mapping of system issues. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, page 31. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[169] M. L. Neal, M. T. Cooling, L. P. Smith, C. T. Thompson, H. M. Sauro, B. E. Carlson, D. L. Cook, and J. H. Gennari. A Reappraisal of How to Build Modular, Reusable Models of Biological Systems. *PLoS Comput Biol*, 10(10):e1003849, 10 2014.

[170] D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 304–319. Springer, 2007.

[171] N. L. Novère, A. Finney, M. Hucka, et al. Minimum information requested in the annotation of biochemical models (MIRIAM). *Nature Biotechnology*, 23(12): 1509–1515, 2005.

[172] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. Artima, 2008.

[173] A. I. of Aeronautics and A. Staf. *AIAA Guide for the Verification and Validation of Computational Fluid Dynamics Simulations*. American Institute of Aeronautics & Astronautics, 1998.

[174] T. I. Ören. GEST—a modelling and simulation language based on system theoretic concepts. In *Simulation and model-based methodologies: an integrative view*, pages 281–335. Springer, 1984.

[175] C. M. Overstreet, R. E. Nance, and O. Balci. Issues in enhancing model reuse. In *International Conference on Grand Challenges for Modeling and Simulation*, pages 27–31, 2002.

[176] G. Özhan, E. Sezgin, D. Wehner, et al. Lypd6 enhances Wnt/β-catenin signaling by promoting Lrp6 phosphorylation in raft plasma membrane domains. *Developmental cell*, 26(4):331–345, 2013.

[177] E. H. Page and J. M. Opper. Observations on the complexity of composable simulation. In *Proceedings of the 31st winter simulation conference*, pages 553–560. ACM, 1999.

[178] E. H. Page, A. Buss, P. A. Fishwick, K. J. Healy, R. E. Nance, and R. J. Paul. Web-based simulation: revolution or evolution? *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 10(1):3–17, 2000.

[179] E. H. Page, L. Litwin, M. T. McMahon, et al. Goal-directed grid-enabled computing for legacy simulations. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 873–879. IEEE, 2012.

[180] O. Pârvu and D. Gilbert. Automatic validation of computational models using pseudo-3d spatio-temporal model checking. *BMC Systems Biology*, 8(1):1–24, 2014.

[181] R. J. Paul and S. J. Taylor. What use is model reuse: Is there a crook at the end of the rainbow? In *Proceedings of the 2002 winter simulation conference*, volume 1, pages 648–652. IEEE, 2002.

[182] K. Pawlikowski, H.-D. Jeong, and J.-S. Lee. On credibility of simulation studies of telecommunication networks. *IEEE Communications Magazine*, 40(1):132–139, 2002.

[183] D. Peng, A. Steiniger, T. Helms, and A. Uhrmacher. Towards Composing ML-Rules Models. In *Proceedings of the 2013 winter simulation conference*, 2013.

[184] D. Peng, R. Ewald, and A. M. Uhrmacher. Towards Semantic Model Composition via Experiments. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, pages 151–162, New York, NY, USA, 2014. ACM.

[185] D. Peng, T. Warnke, and M. A. Uhrmacher. Domain-specific languages for flexibly experimenting with stochastic models. *Simulation Notes Europe*, 25(2):17–22, 2015.

[186] D. Peng, T. Warnke, F. Haack, and A. M. Uhrmacher. Reusing simulation experiment specifications to support developing models by successive extension. *Simulation Modelling Practice and Theory*, 68:33–53, 2016.

[187] L. F. Perrone, C. S. Main, and B. C. Ward. Safe: Simulation automation framework for experiments. In *Proceedings of the 2012 winter simulation conference*, WSC '12, pages 249:1–249:12. Winter Simulation Conference, 2012.

[188] M. D. Petty and E. W. Weisel. A composability lexicon. In *Spring Simulation Interoperability Workshop (SISO)*, pages 181–187, 2003.

[189] M. D. Petty and E. W. Weisel. A formal basis for a theory of semantic composability. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, pages 416–423, 2003.

[190] M. Pidd. Reusing simulation components: simulation software and model reuse: a polemic. In *Proceedings of the 34th winter simulation conference*, pages 772–775. Winter Simulation Conference, 2002.

[191] A. Pos, P. Borst, J. Top, and H. Akkermans. Reusability of simulation models. *Knowledge-Based Systems*, 9(2):119–125, 1996.

[192] R. J. Prill, P. A. Iglesias, and A. Levchenko. Dynamic properties of network motifs contribute to biological network organization. *PLOS Biology*, 3(11):e343, Oct. 2005.

[193] H. Rahmandad and J. D. Sterman. Reporting guidelines for simulation-based research in social sciences. *System Dynamics Review*, 28(4):396–411, 2012.

[194] R. Randhawa, C. a. Shaffer, and J. J. Tyson. Model aggregation: a building-block approach to creating large macromolecular regulatory networks. *Bioinformatics (Oxford, England)*, 25(24):3289–95, Dec. 2009.

[195] R. Randhawa, C. A. Shaffer, and J. J. Tyson. Model Composition for Macromolecular Regulatory Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):278–287, 2010.

[196] E. Renshaw. *Modelling Biological Populations in Space and Time*. Cambridge Studies in Mathematical Biology. Cambridge University Press, 1991.

[197] T. Rharass, H. Lemcke, M. Lantow, et al. Ca2+-mediated mitochondrial ROS metabolism augments Wnt/β-catenin pathway activation to facilitate cell differentiation. *Journal of Biological Chemistry*, 289(40):27937–27951, 2014.

[198] G. Rill. Vehicle modeling by subsystems. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 28(4):430–442, 2006.

[199] S. Robinson. *Simulation: The Practice of Model Development and Use*. Wiley, 2004.

[200] S. Robinson. Conceptual modelling for simulation part i: definition and requirements. *Journal of the operational research society*, 59(3):278–290, 2008.

[201] S. Robinson, R. E. Nance, R. J. Paul, M. Pidd, and S. J. Taylor. Simulation model reuse: definitions, benefits and obstacles. *Simulation modelling practice and theory*, 12(7):479–494, 2004.

[202] S. Robinson, G. Arbez, L. G. Birta, A. Tolk, and G. Wagner. Conceptual modeling: definition, purpose and benefits. In *Proceedings of the 2015 winter simulation conference*, pages 2812–2826. IEEE, 2015.

[203] M. Röhl and A. M. Uhrmacher. Composing simulations from xml-specified model components. In *Proceedings of the 2006 winter simulation conference*, pages 1083–1090. IEEE, 2006.

[204] N. D. Rollins, C. M. Barton, S. Bergin, M. A. Janssen, and A. Lee. A computational model library for publishing model documentation and code. *Environmental Modelling & Software*, 61:59–64, 2014.

[205] A. Ruscheinski. Artefaktbasierte Workflows zur semi-automatischen Validierung von Modellerweiterungen. Master's thesis, University of Rostock, 2016.

[206] S. Rybacki, J. Himmelspach, F. Haack, and A. M. Uhrmacher. WorMS- a Framework to Support Workflows in M&S. In *Proceedings of the 2011 winter simulation conference*, WSC '11, pages 716–727. Winter Simulation Conference, 2011.

[207] S. Rybacki, S. Leye, J. Himmelspach, and A. M. Uhrmacher. Template and frame based experiment workflows in modeling and simulation software with worms. In *2012 IEEE Eighth World Congress on Services*, pages 25–32. IEEE, 2012.

[208] S. Rybacki, F. Haack, K. Wolf, and A. M. Uhrmacher. Developing simulation models-from conceptual to executable model and back-an artifact-based workflow approach. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 21–30. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[209] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

[210] S. Sahle, R. Gauges, J. Pahle, et al. Simulation of Biochemical Networks Using COPASI: A Complex Pathway Simulator. In *Proceedings of the 38th winter simulation conference*, WSC '06, pages 1698–1706. Winter Simulation Conference, 2006.

[211] H. Sakane, H. Yamamoto, and A. Kikuchi. LRP6 is internalized by Dkk1 to suppress its phosphorylation in the lipid raft and is recycled for reuse. *Journal of cell science*, 123(3):360–368, 2010.

[212] S. Sanchez and H. Wan. Work smarter, not harder: A tutorial on designing and conducting simulation experiments. In *Proceedings of the 2012 winter simulation conference*, pages 1–15, Dec 2012.

[213] S. M. Sanchez. Work smarter, not harder: guidelines for designing simulation experiments. In *Proceedings of the 37th winter simulation conference*, pages 69–82. Winter Simulation Conference, 2005.

[214] R. G. Sargent. A tutorial on verification and validation of simulation models. In *Proceedings of the 16th winter simulation conference*, pages 114–121. IEEE Press, 1984.

[215] R. G. Sargent. Verification and validation of simulation models. *Journal of simulation*, 7(1):12–24, 2013.

[216] P. Savory and G. Mackulak. The science of simulation modeling. *Science*, 1:115–119, 1994.

[217] F. Scholkmann, J. Boss, and M. Wolf. An efficient algorithm for automatic peak detection in noisy periodic and quasi-periodic signals. *Algorithms*, 5(4):588–603, 2012.

[218] M. Schulz, J. Uhlendorf, E. Klipp, and W. Liebermeister. SBMLmerge, a system for combining biochemical network models. *Genome Informatics Series*, 17(1):62, 2006.

[219] J. Schützel, D. Peng, A. M. Uhrmacher, and L. F. Perrone. Perspectives on Languages for Specifying Simulation Experiments. In *Proceedings of the 2014 winter simulation conference*, WSC '14, pages 2836–2847, Piscataway, NJ, USA, 2014. IEEE Press.

[220] M. L. Scott. *Programming Language Pragmatics*. Elsevier Science, 2015.

[221] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *International Conference on Computer Aided Verification*, pages 202–215. Springer, 2004.

[222] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In K. Etessami and S. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2005.

[223] K. Sen, M. Viswanathan, and G. Agha. On Statistical Model Checking of Stochastic Systems. In *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer, Berlin Heidelberg, Jan. 2005.

[224] R. E. Shannon. Introduction to the art and science of simulation. In *Winter Simulation Conference*, pages 7–14, 1998.

[225] Y. Shoham. Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence. Technical report, Yale Univ., New Haven, CT (USA), May 1987.

[226] J. Šimunek, M. T. Van Genuchten, and M. Šejna. HYDRUS: Model use, calibration, and validation. *Transactions of the ASABE*, 55(4):1263–1274, 2012.

[227] C. U. Smith, C. M. Llado, R. Puigjaner, and L. G. Williams. Interchange formats for performance models: Experimentation and ouput. In *Quantitative Evaluation of Systems, 2007. QEST 2007. Fourth International Conference on the*, pages 91–100. IEEE, 2007.

[228] C. U. Smith, C. M. Lladó, and R. Puigjaner. Model interchange format specifications for experiments, output and results. *The Computer Journal*, 54(5):674–690, 2011.

[229] L. P. Smith, M. Hucka, S. Hoops, A. Finney, M. Ginkel, C. J. Myers, I. I. Moraru, and W. Liebermeister. Sbml level 3 package specification: Hierarchical model composition. 2013.

[230] M. Spiegel, P. F. Reynolds Jr, and D. C. Brogan. A case study of model context for simulation composability and reusability. In *Proceedings of the 37th winter simulation conference*, pages 437–444. Winter Simulation Conference, 2005.

[231] J. Stelling, P. Mendes, E. Klipp, et al. Defining modeling strategies for Systems Biology. Technical report, FutureSysBio Workshop, Göteborg, Sweden, 2011.

[232] C. Szabo. *Composable simulation models and their formal validation*. PhD thesis, National University of Singapore, 2010.

[233] C. Szabo and Y. M. Teo. On syntactic composability and model reuse. In *First Asia International Conference on Modelling & Simulation, 2007. AMS'07.*, pages 230–237. IEEE, 2007.

[234] C. Szabo and Y. M. Teo. An approach for validation of semantic composability in simulation models. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 3–10. IEEE Computer Society, 2009.

[235] C. Szabo and Y. M. Teo. An approach to semantic-based model discovery and selection. In *Proceedings of the 2011 winter simulation conference*, pages 3054–3066. IEEE, 2011.

[236] C. Szabo and Y. M. Teo. An analysis of the cost of validating semantic composability. *Journal of Simulation*, 6(3):152–163, 2012.

[237] C. Szabo, Y. M. Teo, and S. See. A time-based formalism for the validation of semantic composability. In *Proceedings of the 41st winter simulation conference*, pages 1411–1422. Winter Simulation Conference, 2009.

[238] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-oriented Programming*. ACM Press Series. ACM Press, 2002.

[239] R. J. Tabaczynski, F. H. Trinker, and B. A. Shannon. Further refinement and validation of a turbulent flame propagation model for spark-ignition engines. *Combustion and Flame*, 39(2):111–121, 1980.

[240] S. J. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, J. Zander, and P. J. Mosterman. Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens. *simulation*, page 0037549715590594, 2015.

[241] K. Ten Tusscher and A. Panfilov. Cell model for efficient simulation of wave propagation in human ventricular tissue under normal and pathological conditions. *Physics in medicine and biology*, 51(23):6141, 2006.

[242] Y. M. Teo and C. Szabo. CoDES: An integrated approach to composable modeling and simulation. In *41st Annual Simulation Symposium (anss-41 2008)*, pages 103–110. IEEE, 2008.

[243] A. Teran-Somohano, O. Dayıbaş, L. Yilmaz, and A. Smith. Toward a model-driven engineering framework for reproducible simulation experiment lifecycle management. In *Proceedings of the 2014 winter simulation conference*, pages 2726–2737. IEEE Press, 2014.

[244] A. Teran-Somohano, A. E. Smith, J. Ledet, L. Yilmaz, and H. Oğuztüzün. A model-driven engineering approach to simulation experiment design and execution. In *Proceedings of the 2015 winter simulation conference*, pages 2632–2643. IEEE Press, 2015.

[245] The Sytems Biology Institute. Simulation software for systems biology. `http://systems-biology.org/software/simulation/sbmlsim.html`, 2016.

[246] A. Tolk. What comes after the semantic web - pads implications for the dynamic web. In *Workshop on Principles of Advanced and Distributed Simulation (PADS)*, page 55. IEEE Computer Society, 2006.

[247] A. Tolk, S. Y. Diallo, R. D. King, C. D. Turnitsa, and J. J. Padilla. *Conceptual modeling for composition of model-based complex systems*, pages 355–381. Citeseer, 2010.

[248] T. Tomita, S. Hagihara, and N. Yonezaki. A probabilistic temporal logic with frequency operators and its model checking. In F. Yu and C. Wang, editors, *Proc. of the 13th International Workshop on Verification of Infinite-State Systems, Taipei, Taiwan, 10th October 2011*, volume 73 of *Electronic Proceedings in Theoretical Computer Science*, pages 79–93. Open Publishing Association, 2011.

[249] M. K. Traoré. Analyzing static and temporal properties of simulation models. In *Proceedings of the 38th winter simulation conferencen*, pages 897–904. Winter Simulation Conference, 2006.

[250] T. G. Trucano, L. P. Swiler, T. Igusa, W. L. Oberkampf, and M. Pilch. Calibration, validation, and sensitivity analysis: What's what. *Reliability Engineering & System Safety*, 91(10):1331–1357, 2006.

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

[252] M. T. Vass, C. A. Shaffer, N. Ramakrishnan, L. T. Watson, and J. J. Tyson. The jigcell model builder: a spreadsheet interface for creating biochemical reaction network models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(2):155–164, 2006.

[253] A. Verbraeck and E. C. Valentin. Design guidelines for simulation building blocks. In *Proceedings of the 40th winter simulation conference*, pages 923–932. Winter Simulation Conference, 2008.

[254] V. Vito. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem. R. Accad. Naz. dei Lincei*, 2:31–113, 1926.

[255] D. Waltemath, R. Adams, D. A. Beard, F. T. Bergmann, U. S. Bhalla, R. Britten, V. Chelliah, M. T. Cooling, J. Cooper, E. J. Crampin, et al. Minimum information about a simulation experiment (MIASE). *PLoS computational biology*, 7(4):e1001122, 2011.

[256] D. Waltemath, R. Adams, F. T. Bergmann, et al. Reproducible computational biology experiments with SED-ML - the simulation experiment description markup language. *BMC systems biology*, 5(1):198, 2011.

[257] D. Waltemath, R. Henkel, R. Hälke, M. Scharm, and O. Wolkenhauer. Improving the reuse of computational models through version control. *Bioinformatics*, 29(6): 742–748, 2013.

[258] J. Wang, Q. Chang, G. Xiao, N. Wang, and S. Li. Data driven production modeling and simulation of complex automobile general assembly plant. *Computers in Industry*, 62(7):765–775, 2011.

[259] S. Wang and G. Wainer. Semantic selection for model composition using samsaas. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 25–32. Society for Computer Simulation International, 2015.

[260] T. Warnke, D. Peng, F. Haack, and A. M. Uhrmacher. Towards a language for specifying properties of simulation trajectories. In *12th International Conference on Computational Methods in Systems Biology*, 2014.

[261] T. Warnke, T. Helms, and A. M. Uhrmacher. Syntax and semantics of a multi-level modeling language. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, pages 133–144, New York, NY, USA, 2015. ACM.

[262] E. Weisel, M. Petty, and R. Mielke. Validity of models and classes of models in semantic composability. *Proceedings of the Fall 2003 SIW*, 9:68, 2003.

[263] M. R. Wigan. The fitting, calibration, and validation of simulation models. *Simulation*, 18(5):188–192, 1972.

[264] C.-H. Yeang, H. C. Mak, S. McCuine, C. Workman, T. Jaakkola, and T. Ideker. Validation and refinement of gene-regulatory pathways on a network of physical interactions. *Genome biology*, 6(7):1, 2005.

[265] S. Yoo and M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120, Mar. 2012.

[266] H. L. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification*, pages 223–235. Springer, 2002.

[267] H. L. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368 – 1409, 2006.

[268] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings 14th International Conference on Computer Aided Verification, volume 2404 of LNCS*, pages 223–235. Springer, 2002.

[269] T. Yu, C. M. Lloyd, D. P. Nickerson, M. T. Cooling, A. K. Miller, A. Garny, J. R. Terkildsen, J. Lawson, R. D. Britten, P. J. Hunter, et al. The physiome model repository 2. *Bioinformatics*, 27(5):743–744, 2011.

[270] B. Zeigler. *Theory of Modelling and Simulation.* A Wiley-Interscience Publication. John Wiley, 1976.

[271] B. Zeigler, H. Praehofer, and T. Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, 2000.

[272] B. P. Zeigler and H. S. Sarjoughian. Introduction to devs modeling and simulation with java: Developing component-based simulation models. Technical report, University of Arizona, 2003.

[273] B. P. Zeigler, D. Fulton, P. Hammonds, and J. Nutaro. Framework for M&S based system development and testing in net-centric environment. *ITEA Journal*, 26(3): 21–34, 2005.

[274] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 243–252, New York, NY, USA, 2010. ACM.

Selbständigkeitserklärung

Ich, Danhua Peng, erkläre hiermit, dass ich die vorliegende Dissertation mit dem Titel "Reusing Simulation Experiments for Model Composition and Extension" selbständig, ohne die unerlaubete Hilfe und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.