TU Clausthal

An Architecture for Model Behavior Generation for Multiple Simulators

Dissertation
zur Erlangung des Doktorgrades
der Ingenieurwissenschaften

vorgelegt von
Hendrik Martin Folkerts
aus Itzehoe

genehmigt von der
Fakultät für Mathematik/Informatik und Maschinenbau
der Technischen Universität Clausthal,
Tag der mündlichen Prüfung
26.01.2024

Dekan
Prof. Dr. Jörg P. Müller

Vorsitzender der Promotionskommission
Prof. Dr. Rüdiger Ehlers

Betreuer
Prof. Dr.-Ing. habil. Umut Durak

Gutachter
Prof. Dr. Sven Hartmann

Gutachter
Prof. Dr.-Ing. Thorsten Pawletta, Hochschule Wismar

Diese Arbeit wurde mit LaTeX unter Nutzung von MiKTeX 2.9 als TeX Distribution gesetzt. Als LaTeX Editor wurde das TeXstudio 2.12.4 verwendet. Teile des englischen Textes wurden mit dem DeepL Übersetzer (www.deepl.com) auf die Korrektheit der Sprache geprüft.

Danksagung

Diese Dissertation ist im Rahmen einer kooperativen Promotion mit der Technischen Universität Clausthal an der Hochschule Wismar in der Forschungsgruppe Computational Engineering and Automation (CEA) entstanden.

Auf Seiten der Hochschule Wismar möchte ich insbesondere Prof. Dr.-Ing. Thorsten Pawletta für die interessante Forschung, die umfangreiche Betreuung und stetige Unterstützung danken. Ausdrücklich bedanke ich mich für die viele Zeit, die er für mich aufgewandt hat, die konstruktiven Ratschläge und die fortwährende Motivation bei dieser Arbeit. Außerdem möchte ich mich für die Hilfe beim Aufbau des Kontaktes zu nationalen und internationalen Professoren und Forschungsgruppen bedanken. Über diese habe ich Anregungen erhalten, die zu weiteren Forschungen und Publikationen geführt haben. Nicht vergessen möchte ich an dieser Stelle Prof. Dr.-Ing. Sven Pawletta, der mich im Zuge meiner Masterarbeit im Bereich Elektro- und Informationstechnik der Hochschule Wismar auf das interessante Thema aufmerksam gemacht hat und mich in der Promotionsphase fortwährend unterstützt hat.

Auf Seiten der Technischen Universität Clausthal bedanke ich mich ausdrücklich für die Übernahme der Betreuung durch Prof. Dr.-Ing. habil. Umut Durak, der bei der Bearbeitung des Themas wertvolle Hinweise gegeben hat und den Kontakt zur Technischen Universität Clausthal auch durch mehrere Promovendenseminare unterstützt hat. Weiterhin danke ich Prof. Dr. Sven Hartmann für die Bereitschaft zur Übernahme meiner Betreuung und der Begutachtung der Arbeit.

Auch danke ich allen Mitarbeitern der Forschungsgruppe Computational Engineering and Automation der Hochschule Wismar für die konstruktiven Diskussionen und die gemeinsame Zeit.

Bei meinen Eltern möchte ich mich für die fortwährende Förderung und Unterstützung meines Studiums bedanken. Weiterhin danke ich Gerhard Niehaus für die Tipps und Unterstützung bei englischen Formulierungen. Schließlich bedanke ich mich bei allen, die mich kontinuierlich motiviert haben und mir neue Perspektiven bei der Arbeit und privat aufgezeigt haben.

Abstract

Due to the increasing variety of products, the complexity in the development of technical systems is growing more and more. Accordingly, the management of system variants is a central aspect in the investigation of multifaceted systems with methods of *Modeling and Simulation* (M&S). If the systems under investigation are modular-hierarchical, the *System Entity Structure* (SES)/*Model Base* (MB) framework is an adequate and established approach to variant management in M&S. This approach is based on a strict separation of system variants or system structures on the one hand and reusable and parameterizable basic components on the other hand. Different system structures, system variants, as well as different system parameterizations are called system configurations. While system configurations are represented in an SES, basic components of models are organized in an MB. The SES is a special tree structure and the basic components represent dynamic systems with defined input and output interfaces. The general SES/MB approach describes the SES as an ontology for cross-domain and simulator-independent specification of system configurations. The cross-domain nature of the SES is confirmed by a large number of practical applications. Previous software implementations of the SES/MB framework show that the organization of the MB in this respect is always simulator-specific. Furthermore, implementations of the SES/MB framework also exhibit simulator dependencies in SES, namely in referencing and parameterizing basic components of an MB. These limitations are not of concern if only one specific M&S environment is used. However, this work aims to overcome this limitation and support a model generation independent of an M&S environment.

Complex and multifaceted systems, such as *Cyber-Physical Systems* (CPS), consist of components from different technical domains. Components are modeled and simulated with different and often domain-specific M&S environments. To generate an overall system model, components in the MB must be organized for different M&S environments. Furthermore, the specification of system configurations in the SES must also be *truly* simulator-independent.

For cross-simulator M&S using the SES/MB framework, different methods are developed in this thesis. Two approaches are presented for a simulator-independent organization of an MB. One approach propagates the extension of the MB by a special software component, while the second approach builds on the *Functional Mock-up Interface* (FMI). FMI is an interface standard for models and is widely used in engineering. Furthermore, concepts for simulator-independence of the specification of an SES and for automated selection of configuration variants based on the SES are developed. All approaches are embedded in an extended SES/MB-based software architecture. This supports an automation of simulation studies starting with the selection of system configurations up to the generation and execution of simulation models using different M&S environments.

For the evaluation of the developed methods extensive software implementations were done, which are conceptually pointed out. Furthermore, the application of all concepts is evaluated on the basis of use cases from different domains. The use cases underline the cross-system and cross-domain functionality of the developed methods. Overall, the work contributes to simulator-independent variant management and to the automation of simulation studies of models of modular-hierarchical and multifaceted systems.

Kurzfassung

Durch die steigende Produktvielfalt nimmt die Komplexität bei der Entwicklung technischer Systeme zunehmend zu. Demgemäß ist das Management von Systemvarianten ein zentraler Aspekt bei der Untersuchung vielgestaltiger Systeme mit Methoden der *Modeling and Simulation* (M&S). Sind die zu untersuchenden Systeme modular-hierarchisch aufgebaut, ist das *System Entity Structure* (SES)/*Model Base* (MB)-Framework ein adäquater und etablierter Ansatz zum Variantenmagement in der M&S. Dieser Ansatz basiert auf einer strikten Trennung von Systemvarianten oder Systemstrukturen einerseits und wiederverwendbaren und parametrierbaren Basiskomponenten andererseits. Verschiedene Systemstrukturen, Systemvarianten sowie unterschiedliche Systemparametrierungen werden als Systemkonfigurationen bezeichnet. Während die Systemkonfigurationen in einer SES dargestellt werden, werden Basiskomponenten von Modellen in einer MB organisiert. Die SES ist eine spezielle Baumstruktur und die Basiskomponenten repräsentieren dynamische Systeme mit definierten Ein- und Ausgangsschnittstellen. Der allgemeine SES/MB-Ansatz beschreibt die SES als eine Ontologie zur domänenübergreifenden und simulatorunabhängigen Spezifikation von Systemkonfigurationen. Der domänenübergreifende Charakter der SES wird durch eine Vielzahl praktischer Anwendungen bestätigt. Bisherige softwaretechnische Umsetzungen des SES/MB-Frameworks zeigen, dass die Organisation der MB dahingehend immer simulatorspezifisch erfolgt. Weiterhin weisen Implementationen des SES/MB-Frameworks auch bei der SES Simulatorabhängigkeiten auf, nämlich beim Referenzieren und Parametrieren von Basiskomponenten der MB. Diese Beschränkungen sind nicht von Belang, wenn nur eine spezifische M&S Umgebung zum Einsatz kommt. Diese Arbeit zielt darauf ab, diese Beschränkung zu überwinden und eine von einer M&S-Umgebung unabhängige Modellgenerierung zu unterstützen.

Komplexe und vielgestaltige Systeme, wie zum Beispiel *Cyber-Physical Systems* (CPS), bestehen aus Komponenten verschiedener technischer Domänen. Komponenten werden mit unterschiedlichen und oft domänenspezifischen M&S Umgebungen modelliert und simuliert. Zur Generierung eines Gesamtsystemmodells müssen in der MB Komponenten für verschiedene M&S Umgebungen organisiert werden. Weiterhin muss auch die Spezifikation von Systemkonfigurationen in der SES *echt* simulatorunabhängig sein.

Zur simulatorübergreifenden M&S mit dem SES/MB-Framework werden in der Arbeit verschiedene Methoden entwickelt. Zur simulatorunabhängigen Organisation einer MB werden zwei Ansätze vorgestellt. Ein Ansatz propagiert die Erweiterung der MB um eine spezielle Softwarekomponente, während der zweite Ansatz auf das *Functional Mock-up Interface* (FMI) aufbaut. FMI ist ein Schnittstellenstandard für Modelle und ist im Ingenieursbereich stark verbreitet. Weiterhin werden Konzepte zur Simulatorunabhängigkeit der Spezifikation einer SES und zur automatisierten Auswahl von Konfigurationsvarianten auf Basis der SES entwickelt. Alle Ansätze werden in eine erweiterte SES/MB-basierte Soft-

warearchitektur eingebettet. Diese unterstützt eine Automation von Simulationsstudien beginnend bei der Auswahl von Systemkonfigurationen bis zur Generierung und Ausführung von Simulationsmodellen unter Nutzung verschiedener M&S Umgebungen. Zur Evaluierung der entwickelten Methoden erfolgten umfangreiche softwaretechnische Umsetzungen, die konzeptionell aufgezeigt werden. Weiterhin wird die Anwendung aller Konzepte anhand von Use-Cases aus verschiedenen Domänen gezeigt und evaluiert. Die Use-Cases unterstreichen die system- und domänenübergreifende Funktionalität der entwickelten Methoden. Insgesamt wird mit der Arbeit ein Beitrag zum simulatorunabhängigen Variantenmanagement und zur Automation von Simulationsstudien von Modellen modular-hierarchischer und vielgestaltiger Systeme geleistet.

Contents

Acronyms

API	Application Programming Interface
ASIM	Arbeitsgemeinschaft Simulation
BPMN	Business Process Model and Notation
CEA	Computational Engineering and Automation
CIM	Computation-independent Model
CPS	Cyber-Physical System
CSS	Cascading Style Sheets
CSSL	Continuous System Simulation Language
DAM	Downward Atomic Model
DES	Discrete Event System
DESS	Differential Equation System Specification
DEVS	Discrete Event System Specification
DFG	Deutsche Forschungsgemeinschaft
DLL	Dynamic Link Library
DSEEP	Distributed Simulation Engineering and Execution Process
DSL	Domain-Specific Language
DTSS	Discrete Time System Specification
EC	Experiment Control
ECU	Electronic Control Unit
EF	Experimental Frame
EIC	External Input Coupling
EMB	Experiment Method Base
EOC	External Output Coupling
ESE	Executable Simulation-based Experiment
EU	Execution Unit
ExM	Experiment Method
FM	Feature Model
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FoM	Family of Models
FPES	Flattened Pruned Entity Structure
FSM	Finite State Machines
fUML	Foundational UML
GPL	General Purpose Language
GPSS	General Purpose Simulation System
GUI	Graphical User Interface
HPC	High Performance Cluster

HTML	Hypertext Markup Language
IC	Inner Coupling
JSON	Java Script Object Notation
M2M	Model To Model
M2T	Model To Text
MB	Model Base
MBSE	Model-Based Systems Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MES	Model Engineering for Simulation
MOF	Meta-Object Facility
M&S	Modeling and Simulation
MSS	Multicomponent System Specification
MUS	Model Under Study
NumRep	Number of Replications
OCL	Object Constraint Language
ODE	Ordinary Differential Equation
OMG	Object Management Group
OS	Operating System
OWL	Web Ontology Language
PDEVS	Parallel Discrete Event System Specification
PES	Pruned Entity Structure
PIM	Platform-independent Model
PPEA	Physical Port Electrical Analog
PSM	Platform-specific Model
QSS	Quantized State System
RCP	Rapid Control Prototyping
SBE	Simulation-based Experiment
SDK	Software Development Kit
SES	System Entity Structure
SESfcn	SES function
SESvar	SES variable
SM	Simulation Model
SME	Simulation Model Executable
SMR	Simulation Model Representation
SnM	Simulation Method
SPL	Software Product Line
SPLE	Software Product Line Engineering
SPR	Signal Port Real
SSD	System Structure Definition
SSP	System Structure & Parameterization
SysML	Systems Modeling Language
UAM	Upward Atomic Model
UML	Unified Modeling Language
VV&A	Verification, Validation, and Accreditation
W3C	World Wide Web Consortium
XML	Extensible Markup Language

xUML Executable UML

1 Introduction

Modeling and Simulation (M&S) is an established method for the prediction and analysis of the dynamic behavior of systems. In the research of modular-hierarchical and multifaceted systems, variability modeling with suitable methods represents a central aspect. Moreover, the M&S of complex technical systems is often performed using different environments due to domain diversity. Based on these assumptions, first the motivation of this work is detailed and on the basis of a general classification of M&S approaches, the focus of this work is introduced. Finally, the objectives and the structure of the thesis are specified.

1.1 Motivation

In product development, product diversity is constantly increasing. In addition to new products, different variants of a product are also being offered. This is particularly evident in an example from the automotive sector: As early as in the year 2011, Oster [105] writes that some *Electronic Control Unit*s (ECUs) in a car have up to 10,000 different configuration options and more than 50 ECUs are installed. The number of ECUs in a car has been growing ever since. Due to different configurations of the cars, a multitude of individual software variants must be generated for the ECUs. Often, the required functionality for the ECUs is modeled, tested simulatively and then the software for the ECUs is automatically generated from the models. Consequently, the variability of the software for the ECUs must be mapped in the models. Variant management methods are used to present the variants in a structured and clear manner and to facilitate the generation of the software. The structured approach allows the increasing complexity of products to be taken into account. In the following, the umbrella term *technical system* is used synonymously for a technical product in order to emphasize the interdisciplinary relevance of the methods presented in this work.

A central building block in system development is the mapping of the system behavior in models and the subsequent simulation. Complex systems are usually formed from several subcomponents. Different product variants represent systems consisting of different subcomponents with different arrangements. The simulation of systems of different domains is usually done with different simulators, which are usually specialized on the challenges of one domain.

Technical systems consisting of mechanical and electrical components combined with software components and a communication infrastructure are today referred to as *Cyber-Physical System*s (CPS). They are characterized by a high degree of complexity. Modern cars and machine tools are typical examples for CPS. The components of a CPS are often

developed using different domain-specific modeling approaches and modeling software. Thus the simulation of the whole system is a multidisciplinary task.

To keep development costs low, including the necessary M&S for complex and variable systems, approaches of variability modeling have to be combined with concepts of model exchange. Various approaches have been developed for variability modeling in different fields. Examples are the 150% model approach (Alt [2]) or the *System Entity Structure* (SES)/*Model Base* (MB) approach (Zeigler et al. [167]), which are discussed in detail in this thesis. One concept of model exchange is the separation of modeling language and simulation programs. An example is the standardized modeling language Modelica [91]. Different simulation programs can interpret and execute models based on the Modelica standard. An alternative approach is to specify a universal interface, such as the *Functional Mock-up Interface* (FMI) (Blochwitz et al. [13]). Components specified in this way can be organized in an MB and used and executed in different simulation programs.

The system behavior of complex systems can change completely by modifying subcomponents. Therefore, it is necessary to create and test a model for *each* system variant. To minimize the development costs of new variants of a product, extensive automation of model-based testing of system variants is desirable. Automation includes aspects of modeling, test procedure specification, and execution. For this purpose, models for different target simulators should be automatically generatable and executable. The generation of executable models for different target simulators enables the verification of the numerical correctness of simulation programs. Simulators use different implementations of numerical algorithms, even if the same name is used for the simulation algorithm (Junghanns and Blochwitz [71]). The simulation results of one model simulated with various simulators using one type of solver should ideally be identical.

In order to increase the acceptance of a methodical approach among engineers, an important aspect is to provide comprehensive software support. The software must be intuitive to use and provide extensive help, such as automatic checks, design patterns, or examples. Furthermore, the software should provide at least one interface to a standardized data format (e.g. *Java Script Object Notation* (JSON) or *Extensible Markup Language* (XML)) to allow integration into a possibly existing architecture. Last but not least, the software should be easy to install. This ensures the applicability of the developed methods and favors the combination with existing programs. Application by different engineers in different fields and domains ensures comprehensive testing and enables further development.

1.2 General M&S Background

Systems increasingly consist of subcomponents of different domains, while simulation tools are usually specialized on one domain. Consequently, M&S is often performed with different tools. In the following, a classification of M&S approaches in terms of the number of different tools is given.

The cassification in Figure 1.1 according to Geimer et al. [57] focuses on primarily continuous systems. The number of modeling tools is plotted against the number of integrators, which can be equated with simulators in general. The y-axis shows the number

of the used modeling tools, while the x-axis shows the number of different simulators.

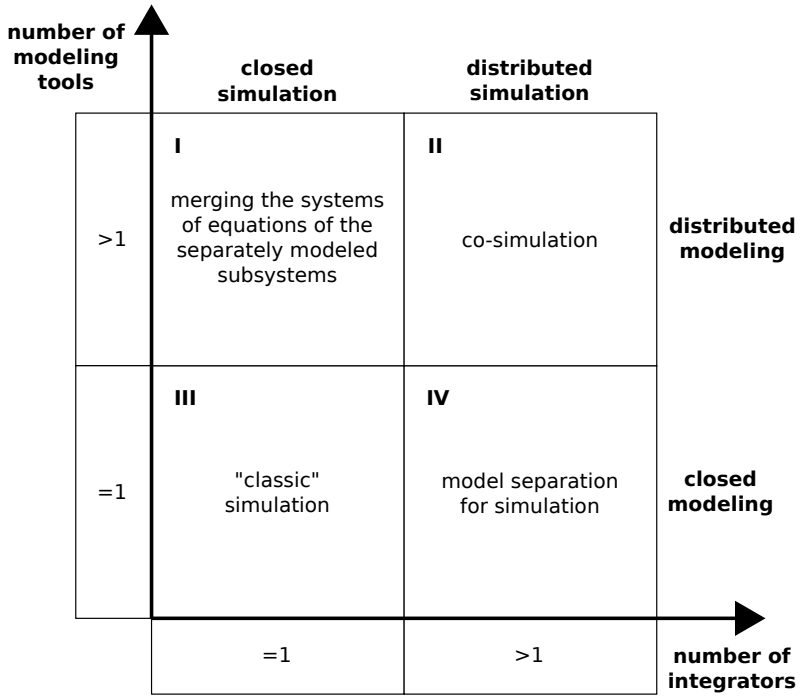


Figure 1.1: Classification of M&S approaches according to Geimer et al. [57] (translated from German to English).

Class III in Figure 1.1 specifies the use of one modeling tool and one simulator, which is called classical simulation. For a multi-physical system, such as a CPS, in this case the M&S environment should support different modeling formalisms. Practical examples are the Modelica language [91] and associated simulators like OpenModelica [103] or Dymola [38], MATLAB/Simulink [87], or AnyLogic [35]. A theoretical formalism concerning this matter is provided by the *Discrete Event System Specification* (DEVS) theory and its extensions (Zeigler et al. [167]).

Class I considers the use of different domain-specific modeling tools, whose models are composed into an overall model and processed with a simulator. For this purpose, a variety of model transformations have been developed that not only support the merging of equation systems. As a representative example, the diverse work of Vangheluwe and colleagues should be mentioned here (Vangheluwe [153], Vangheluwe et al. [154]). Vangheluwe [152] shows that the DEVS theory can be seen as a common denominator for multi-formalism modeling and refers to it as the assembler of modeling paradigms.

Class IV represents the use of one modeling tool and several simulators. The model specification is separated for execution on multiple simulators. The typical use case of this class are approaches for distributed or parallel simulation with the objective of runtime reduction (Fujimoto [56], Fujimoto [55]).

Class II considers the use of different domain-specific M&S tools. Models of subsystems of a whole system are only available for specific simulation tools. Each simulation tool covers a specific domain and suits best for problems of this domain. The challenge of this approach is the coupling of the different simulation tools and the associated numerical problems, analogous to the requirements for distributed simulations according to class

IV, although here the focus is usually not on runtime reduction. Practically implemented simulator couplings are specific solutions for the involved simulators (Bastian et al. [8]). Gomes et al. [59] give a detailed overview of the basics and requirements of this class. In addition to the M&S of multi-physical systems, co-simulation is used in the context of hardware in the loop simulation.

The focus of this work lies in the consideration of M&S approaches of class I and class III. With respect to the M&S of multi-physical systems using different domain-specific modeling approaches, it can be stated for both cases that the execution of the overall multi-physical model by a simulator depends on the model transformation algorithms and the numerical simulation algorithms. The fundamentals of this research are considered in Chapter 2.

Variability modeling is a basic element in today's system development. One approach of variability modeling in M&S is the SES/MB approach. Figure 1.2 shows the SES/MB approach according to Zeigler et al. [167]. For the description of systems a separation into the structures and dynamic components takes place. While the dynamic basic components are organized in an MB, the structure and parameter variants are described with an SES. The SES is a special tree structure. Furthermore, the SES/MB framework defines two basic methods: (i) pruning to derive a specific system variant from the SES and (ii) build to generate an executable model using the components from the MB.

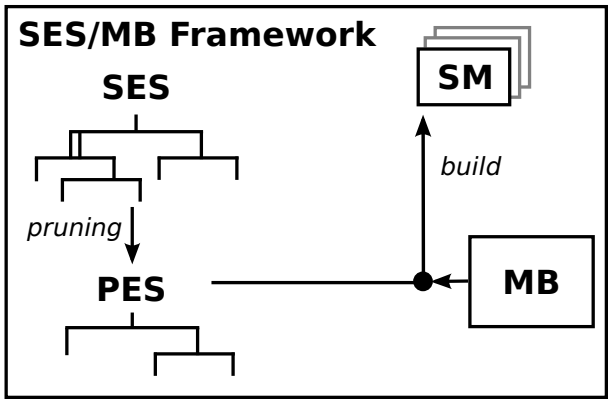


Figure 1.2: The classic SES/MB framework according to Zeigler et al. [167].

1.3 Objectives and Structure of the Thesis

The SES supports a largely simulator-independent specification of system structures and system variants. The modeling and organization of dynamic basic systems in an MB is in this respect usually simulator specific. Accordingly, the derivation of a system variant is simulator-independent and the model generation is specific to an M&S environment. A current requirement is to generate and execute models for different simulators as automatically as possible. The goal of this work is to analyze and further develop the SES/MB approach with respect to: (i) variant management, (ii) automated variant selection, (iii) model generation for different simulators, and (iv) automation of simulation experiments with different model structures and model parameterizations.

Starting from Schmidt [135], the structure of simulation-based experiments is investigated

and it is built on the SES/MB architecture presented there. The approach of Schmidt [135] refers to only one specific M&S environment. Furthermore, not all description means of the SES are supported. In this work, existing limitations regarding modeling with the SES in conjunction with automated variant derivation and generation are addressed by newly developed approaches.

In contrast to the SES, the MB is closely related to the used M&S environment. To use the SES/MB approach across simulators, in this work on the one hand language and simulator specific features are used. This is called *native approach*. On the other hand, the established FMI standard is used as a universal model interface to support cross-simulator model generation. This is called *FMI-based approach*.

Also based on the SES/MB architecture in Schmidt [135], a software architecture for the automation of simulation-based experiments using different M&S environments was developed for both the native approach and the FMI-based approach. The newly developed architecture was implemented prototypically in the form of Python-based tools and using standardized software interfaces and serves as a proof of concept.

The motivation and objectives lead to the investigation of the following hypotheses in this thesis:

1. Modeling variability is a key aspect when analyzing technical systems.
2. Technical systems often need to be tested using different simulators.
3. Combining approaches for variability modeling and model exchange to tackle the challenges in hypothesis 1 and 2.
4. An architecture for the automation of modeling complex systems with high degree of variability and model generation as well as execution using different simulators is useful.
5. The availability of intuitive software tools is the key to introduce new approaches accepted by engineers.

In the following paragraphs, the structure of the thesis is presented, briefly summarizing the contents of each chapter.

Chapter 2 introduces essential aspects of *Model Engineering for Simulation* (MES). Based on a newly introduced life cycle model of M&S, the contents of this thesis are delineated. In this context, a strict distinction is made between the modeling of a dynamic system and its execution in a simulator. Furthermore, different types of dynamic systems are discussed and the connection of M&S to the concept of *Model-Driven Engineering* (MDE) of software engineering is established. The transfer of the objectives of MDE to problems of M&S requires the use of general model interfaces. In the following, FMI is presented as such an interface. In the following, approaches of variant modeling, variant selection, and generation of executable simulation models are presented. For this purpose, the example of a control system is introduced, which will be reused in the following chapters. Finally, the SES/MB approach used in this thesis is considered in the context of MDE.

Chapter 3 details the SES/MB approach in the context of variant management. Extensions of the SES and methods for automated variant selection and model generation

using these extensions are presented. Already known extensions and methods are modified and new extensions are introduced. Subsequently, it is shown how simulation-based experiments can be described and automated using the SES/MB approach.

In *Chapter 4*, shortcomings of an existing SES/MB-based architecture for automated simulation-based experiments are identified. Based on this, a new adapted SES/MB-based software architecture is discussed, with which simulation-based experiments can be described and executed using different target simulators. The components of the software architecture are discussed in detail and illustrated using the example system introduced in Chapter 2. With respect to model generation, a distinction is made between a simulator native and an FMI-based approach. Furthermore, it is shown that the new architecture is also applicable for non-simulation specific applications.

Chapter 5 is dedicated to the proof of concept for the developed methods and the software architecture. Use cases for different system classes and different simulators are presented. Model generation is considered according to both the native approach and the FMI-based approach.

Chapter 6 summarizes key aspects and findings and provides an outlook for possible further work.

2 Model Engineering for Simulation

In this chapter, aspects of *Model Engineering for Simulation* (MES) are discussed. According to Zhang et al. [171] the MES includes the methods involved in the full life cycle of a model for simulation. According to Zeigler et al. [170] the MES defines a model as an abstract expression of objects of the real or imaginary world, the source system, and simulation as a model-based technology to imitate the behavior of objects over time. Accordingly, the full life cycle of a model for simulation is considered first, with an extension to a *Family of Models* (FoM). A FoM is according to Zeigler et al. [167] a set of alternative models from which a candidate model can be selected. Subsequently, some aspects of the full life cycle of a FoM for simulation will be discussed in more detail. It starts with the basic interrelationship between formal modeling of dynamic behavior and the execution of a formal model using a simulator algorithm. Basic system formalisms and modeling paradigms are discussed. Afterwards, the concept of *Model-Driven Engineering* (MDE) is considered in the context of *Modeling and Simulation* (M&S) and the problem of general model interfaces is discussed using the example of the *Functional Mock-up Interface* (FMI). Subsequently, approaches for modeling model variants of a FoM, the selection of candidate models and the generation of executable simulation models are discussed. Finally, essential findings are summarized and the following work steps are roughly derived.

2.1 Life Cycle of a Family of Models for Simulation

In the M&S community different approaches for the life cycle of a model for simulation were developed. Representative approaches are those of Balci [5], Zhang et al. [171], the *Distributed Simulation Engineering and Execution Process* (DSEEP) standard, and the process model of the Working Group *Arbeitsgemeinschaft Simulation* (ASIM) according to Rabe et al. [122]. The approaches reflect essentially identical content. The individual steps and their results are summarized or detailed differently. Schmidt [135] extends the classic approaches to a FoM, based on the process model of the ASIM. Schmidt defines a FoM as a set of models or submodels that share common characteristics. For the purpose of this thesis, the initial definition of a FoM by Zeigler et al. [167] as well as that by Schmidt [135] can be unified and extended to:

A *FoM* is a set of models or submodels that share common characteristics, serve at least partially common objectives, and from which a candidate model can be selected.

Based on the previously listed work, the life cycle of a FoM for simulation is derived according to Figure 2.1.

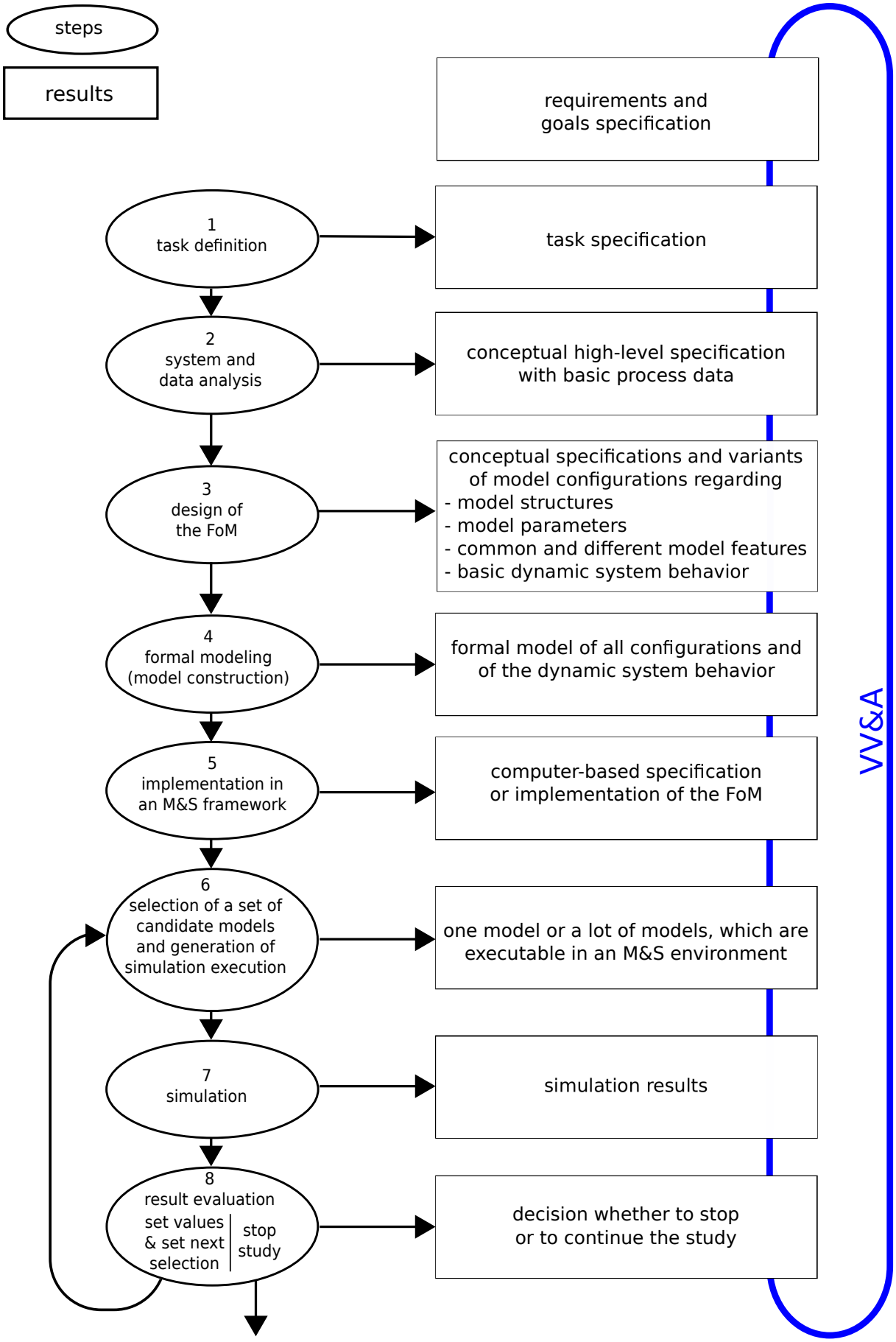


Figure 2.1: Life cycle of a FoM for simulation (developed based on Zhang et al. [171], the ASIM process model according to Rabe et al. [122], and Schmidt [135]).

Analogous to the process model of the ASIM, Figure 2.1 shows the individual steps and their results. Each result on the right side is subject to the *Verification, Validation, and Accreditation* (VV&A) process. The cycles resulting from the VV&A process are not explicitly shown. Regarding details on VV&A, please refer to the extensive literature on VV&A, such as in Masys et al. [85], Petty [115], or Chew and Sullivan [32]. At this point, only some of the special features of the extension to FoM will be discussed. Step 3 already includes variant modeling, which requires variant management. With the objective of a clear modeling and the production of well maintainable models, paradigms of the M&S, like the separation of model structures, model parameters, and dynamic behavior should be considered already in the step 3 consistently. Step 5 was generalized compared to the ASIM process model. The result of the implementation in an M&S framework does not necessarily have to be executable models. It can also be model specifications from which executable code is generated. Step 6 involves variant selection and generation of one or more candidate models that are executable in an M&S environment. The loop above steps 6, 7, and 8 symbolizes the (semi)automated, goal-directed generation, execution and evaluation of simulations.

Like the VV&A, steps 1 and 2 will not be considered in detail. The focus of this work is on aspects of steps 3 to 8. These are the variant modeling and implementation associated with a FoM and the goal-directed selection, generation, and execution (simulation) of candidate models.

Based on the different types of dynamic system behavior, the next section first discusses the formal modeling of dynamic behavior (step 4) and the execution by a simulator. The use of modern M&S frameworks often obscures the fundamental relationship between formal model and simulator. As a result, the user is often not aware of essential aspects of step 4.

2.2 Dynamic System Behavior, System Formalisms, and Modeling Approaches

In the beginning, based on Zeigler et al. [170] a model was defined as an abstract expression of objects of the real or imaginary world and the world was called the source system. Further it was said that with the simulation the behavior of objects over time is examined. Accordingly, in step 4 according to Figure 2.2 the dynamic behavior of a system is to be formally represented in a model. As shown in Figure 2.2, the system to be modeled always represents only a section of a larger world (Bossel [17], Zeigler et al. [167], Zeigler et al. [170], and many others).

The interactions with the environment are modeled with a set of time-dependent influence quantities X and output quantities Y. The dynamics of the system under consideration is modeled with a set of time-dependent state variables and a set of functions. The functions define relations between the sets X, Y, and S. A detailed formal specification of the system elements and the relations is called a system specification according to Zeigler [162]. A formal system specification is the basis for implementing the dynamic behavior of a system on a computer (step 5 in Figure 2.1). With respect to the temporal change of state, four fundamental dynamic behaviors are distinguished as shown in Figure 2.3 (Zeigler

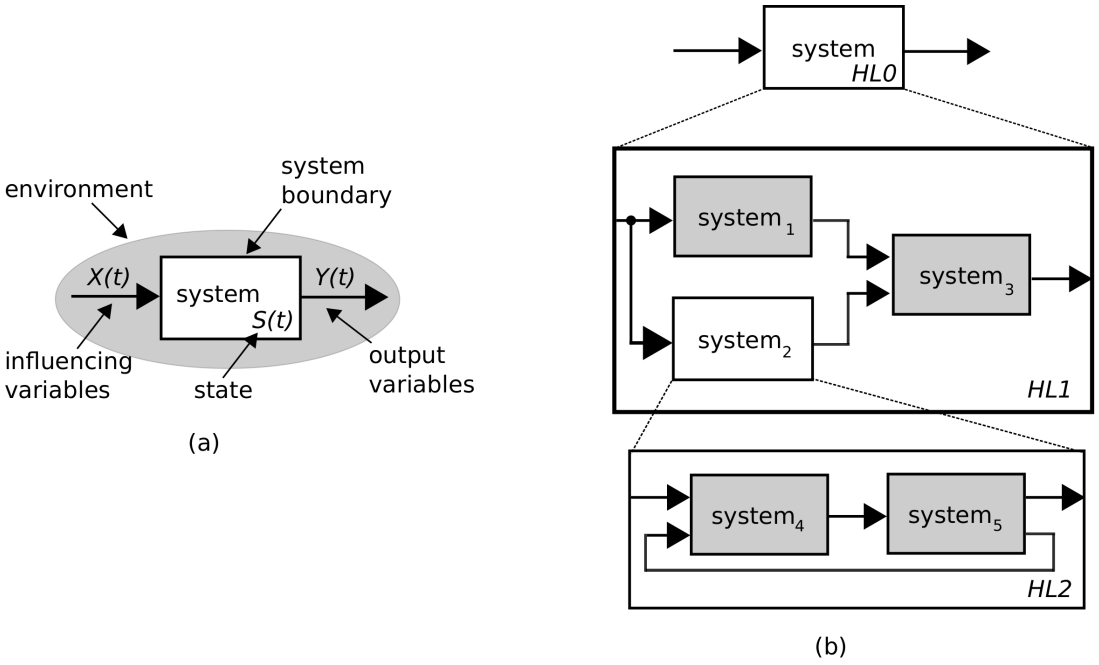


Figure 2.2: (a) A dynamic system as part of a world and (b) hierachical system decomposition according to Schmidt [135], based on Bossel [17] and Zeigler et al. [167].

[162], Zeigler et al. [167], Prähofer [118], Ptolemaeus [119], Lee and Seshia [82], and many others):

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

According to Prähofer [118] and Zeigler et al. [167], all forms of hybrid system dynamics can be described by combining the four basic system specifications. As an example DEV&DESS for the description of combined continuous and discrete event system behavior is mentioned. The execution of a system specification on a computer requires an appropriate simulator algorithm. The integration of a system specification and an appropriate execution algorithm is called a system formalism. In Prähofer [118], Zeigler et al. [167], and Zeigler et al. [170] a comprehensive presentation of the different system specifications and associated simulator algorithms is given. Based on the DEVS formalism, Kofman and Junco (Kofman and Junco [77], Zeigler et al. [170]) introduced the *Quantized State System* (QSS) formalism as an alternative to the DESS formalism for describing continuous systems behavior.

As already mentioned, a FoM specifies a set of alternative models from which a candidate model can be selected. Accordingly, a FoM should be modeled modularly according to the *Multicomponent System Specification* (MSS) (Zeigler et al. [167]) and not as a monothic system. The MSS models a system as a collection of interacting (sub)systems, which may be structured in a hierarchical manner as shown in Figure 2.2 (b). The MSS distinguishes between atomic and coupled systems. The latter are also called networks. Atomic systems define dynamic system behavior based on the basic system specifications or their combinations. As an example, the dynamic specification of an atomic DEVS system is

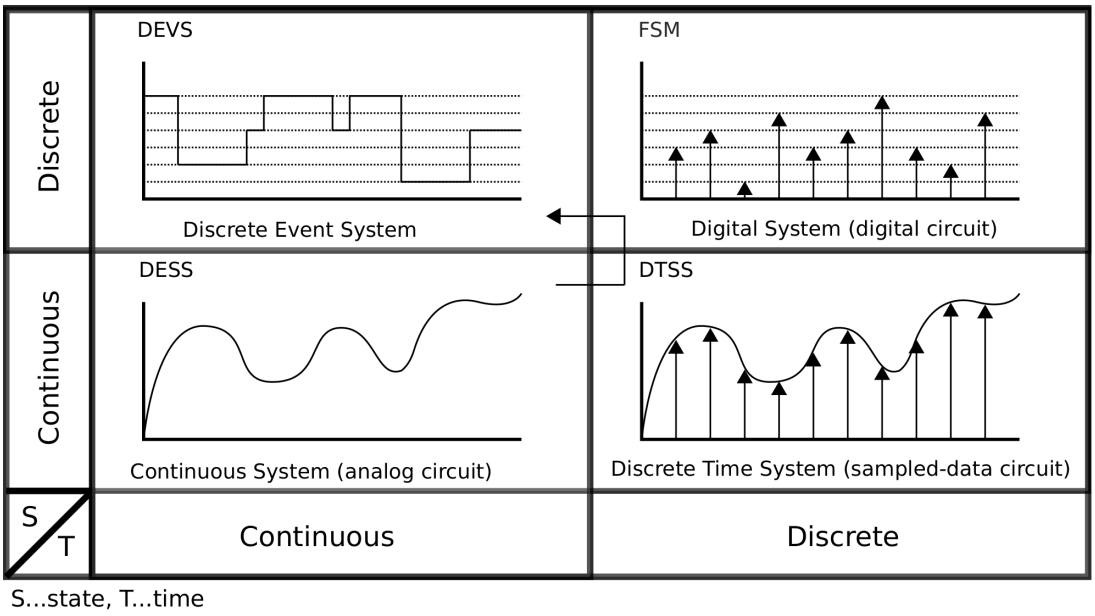


Figure 2.3: System dynamics paradigms according to Prähofer [118].

given (Zeigler et al. [167]). The individual elements of the atomic DEVS specification are explained in Appendix A.

A coupled system defines components, which may be atomic or coupled systems, and interface mappings between the components as well as with the input and the output value set. In the case of directly connected output and input ports the system specification of a coupled system can be simplified (Zeigler et al. [167]). The individual elements of the coupled system specification are explained in Appendix A.

According to Vangheluwe [152] the system formalisms are a kind of assembly language, which more abstract M&S approaches can be mapped on. Already in Zeigler [162] it is shown how abstract modeling views, such as the activity scanning approach, the process interaction approach, and the transaction-oriented method or equation-based models, can be mapped to the system formalisms. The same is true for other abstract M&S paradigms such as statecharts (Borland and Vangheluwe [16], Risco-Martín et al. [124]) or signal flow-oriented methods (Cellier and Kofman [28]), and many others such as Petri-Nets, physical modeling approaches, or VHDL specifications as shown by Wainer [156]. Depending on the application domain, different modeling paradigms are used. Modeling paradigms are often implemented as *Domain-Specific Language* (DSL). One of the first DSL in the field of M&S was the *General Purpose Simulation System* (GPSS) developed by Gordon in the beginning of the 1960s (Gordon [60]). GPSS is based on the transaction-oriented world view and was designed in particular for operation research problems. However, mapping a modeling paradigm to a system formalism is often not transparent in today's DSL or M&S software environments. Often this results in non-theory conform implementations.

M&S methods are an integral part of systems engineering of complex systems. Systems engineering is increasingly based on the MDE approach. Accordingly, the next section discusses interrelationships between the M&S and MDE.

2.3 Model-Driven Engineering and its Relations to the M&S Domain

Complex systems, as characterized in Chapter 1, consist of multiple components of different domains that are closely interrelated. A typical example of such systems are *Cyber-Physical Systems* (CPS). A CPS is composed of mechanical, electrical, and other technical components, as well as components from information technology (CPS Steering Group [37], Lee and Seshia [82], Sanislav and Miclea [130]). The development and realization of such complex systems is a technological challenge, which requires more than just a procedure model, such as the V-model [42]. At relatively the same time, the continuous use of models from requirements specification to system testing and operation was seen as a solution approach for systems engineering of complex systems. In this context, it is important to reuse the models in the individual development phases and to refine them step by step.

Building on foundational work in systems theory, Wymore [161] developed the general *Model-Based Systems Engineering* (MBSE) approach. In the field of automation technology, the concept of *Rapid Control Prototyping* (RCP) was developed (Abel and Bollig [1]). The software engineering community developed the idea of MDE based on the Computer-Aided Software Engineering from the 1980s.

2.3.1 Introduction to Model-Driven Engineering and the Model-Driven Architecture

MDE is a methodology that focuses on the use of models as primary elements of system development (Schmidt [138]). From the MDE approach, the *Model-Driven Development* (MDD) method was derived, which further increases the use of models in software development by using models to drive the entire development process (Cetinkaya et al. [29]). In the context of this work, no further distinction is made between MDE and MDD. Both approaches are based on the same basic principles and pursue analogous objectives. By refinement of models and by means of transformation methods, final executable code for the operation phase is to be generated as automatically as possible on the basis of source models from the requirements specification phase and the conceptual modeling phase. The implementation of this objective is based on the basic pattern: meta-metamodel, metamodel and model according to Figure 2.4.

The MDE approach is specified by different standards. The *Model-Driven Architecture* (MDA) [101] provides the most general specification of MDE developed by the *Object Management Group* (OMG) since 2001. It is an MDE approach commonly used in software engineering. At the center of MDA is a model, and with the help of transformation methods new models are created. Thus the key principle of MDE of recursive models is fulfilled ("Everything is a model") (Brambilla et al. [18], Brambilla et al. [19]). The aim of MDA is the separation of functionality and the realization on a specific software platform. For this purpose, MDA defines three levels of abstraction for models.

- *Computation-independent Model (CIM)*: it focuses on the requirements of a system and its interrelations with the environment.

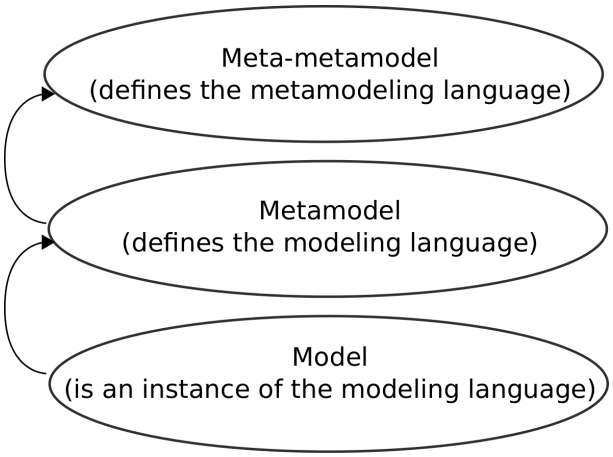


Figure 2.4: Metamodeling pattern in MDE according to Cetinkaya et al. [29].

- *Platform-independent Model (PIM)*: it describes the system structure and its behavior based on the CIM, and independent from the later realization on a specific platform.
- *Platform-specific Model (PSM)*: it includes technical details for execution on a specific platform.

The concept of MDA is supported by several other OMG standards. One essential element is the *Unified Modeling Language* (UML) [99]. UML is a *General Purpose Language* (GPL) and was primarily developed as a modeling language for software systems. The different diagram types of the UML support modeling on CIM level as well as on PIM level. To support the transformation of PIM into PSM, the *Executable UML* (xUML) was developed based on a subset of the UML (Mellor and Balcer [89]), which defines an operational semantics. The *Foundational UML* (fUML), which is favored today, was developed on the basis of xUML [102].

The UML can be used to model technical systems, but its focus is on modeling software systems. The UML was adapted for systems engineering of technical systems to the *Systems Modeling Language* (SysML) (Alt [2]). Concepts were added, which are necessary for describing technical systems, specific parts necessary for modeling of software systems are left away. Since physical systems follow the laws of nature, a new diagram type was introduced, called Parametric Diagram. In the Parametric Diagram constraints of a system can be defined and thus the semantics of model elements can be limited (Weilkiens [158], [141]). Another diagram type introduced in SysML is the Requirement Diagram. In this diagram requirements for a system and relationships between model elements are described. Other adaptations in SysML regard further differences between software systems and physical systems.

2.3.2 The Application of Model-Driven Engineering and the Model-Driven Architecture in the Field of M&S

The application of MDE principles and the MDA specification in the M&S domain has already been studied in different works. A relatively recent overview of the state of the art is provided by Durak [44] or Topçu et al. [145]. They see the application of

the concept of MDE in simulation engineering in the generation of simulation software code via the transformation and refinement of models. Their specific goals in using MDE principles include improving interoperability between distributed simulation systems, reverse engineering, and modernizing simulation systems. Accordingly, they consider the transformation from a DSL to a GPL, namely the transformation of a Simulink model to SysML.

Durak et al. [45] relate the development of an ontology for trajectory simulation to MDA. The use of MDE principles according to the steps of the life cycle of a model in the M&S domain is considered for example by Cetinkaya et al. [30] as well as by Heinzl [68]. In both papers, a prototype of a software framework according to the MDA specification is described. In Cetinkaya et al. [30] the conceptual modeling is done on the CIM level using the *Business Process Model and Notation* (BPMN). The BPMN model is automatically transformed into DEVS, a formal model on the PIM level. Necessary refinement steps are not discussed. The BPMN to DEVS transformation is shown in Cetinkaya et al. [30]. Then, the formal DEVS model, the PIM, is transformed into a PSM using a specific DEVS simulation library. Cetinkaya et al. [30] analyze several other works dealing with the transformation from UML to DEVS. The approach of Heinzl [68] is similar in principle to that of Cetinkaya et al. [30]. For conceptual modeling, Heinzl uses a specially developed modeling language called Cubes. Using a *Model To Model* (M2M) transformation, a PIM is generated from the Cubes-based CIM. The basis of the PIM is an extended DEVS specification for hybrid systems, which is then translated via *Model To Text* (M2T) transformation using a special DEVS library [150] into a MATLAB-based PSM.

As discussed in the previous subsection, systems engineering of physical systems primarily uses SysML instead of UML. Kapos et al. [74] and Tsadimas et al. [147] discuss MDA approaches for transforming SysML models into executable DEVS models. Further works are analyzed by Blas et al. [12], that address the transformation of executable simulation models based on platform-independent, formal DEVS models. However, according to Blas et al., the actually platform-independent formal modeling based on DEVS is often influenced by special circumstances of the implementation level, i.e. the M&S environment for which executable models are to be generated. For this reason, a universal representation of DEVS models in the form of a metamodel-based definition of the DEVS system specification using UML and the *Object Constraint Language* (OCL) is proposed by Blas et al.. The universal representation is seen as the basis for M2M transformations into executable, platform-dependent DEVS models for different M&S environments. It has to be mentioned that the universal representation is only valid for Classic DEVS with ports according to Zeigler et al. [167].

In systems engineering of physical systems, like CPS, there is a crucial difference to the work analyzed before. A CPS consists of complex subsystems of different technical domains. According to Bertram [11], the SysML is used as a cross-domain GPL in the context of requirements specification and conceptual modeling to create a shared system model at CIM level. Some subsystems are also detailed down to PIM level using SysML. M&S in the domain of individual technical spaces is done with DSLs, as shown schematically in Figure 2.5. The DSLs have often been established in the respective technical spaces for decades and have only been replaced to a limited extent by multiphysical DSLs, such as Modelica [91]. Model transformations and model refinements from the SysML-based system model

into the DSLs of the technical spaces are mostly done manually. Furthermore, in practice there are direct interrelations between models of the technical spaces, indicated by the dashed lines. As shown in Figure 2.5, the implementation of the MDE principles by means of metamodels and model transformations is extremely complex due to the large number of DSLs involved in the development process, especially since the construction of an overall simulation requires bidirectional transformations between the GPL and the DSLs or, in general, transformations to a system formalism as DEVS. An approach for the translation of SysML models into the discrete event simulation software Arena [125] is presented by Batarseh and McGinnis [9].

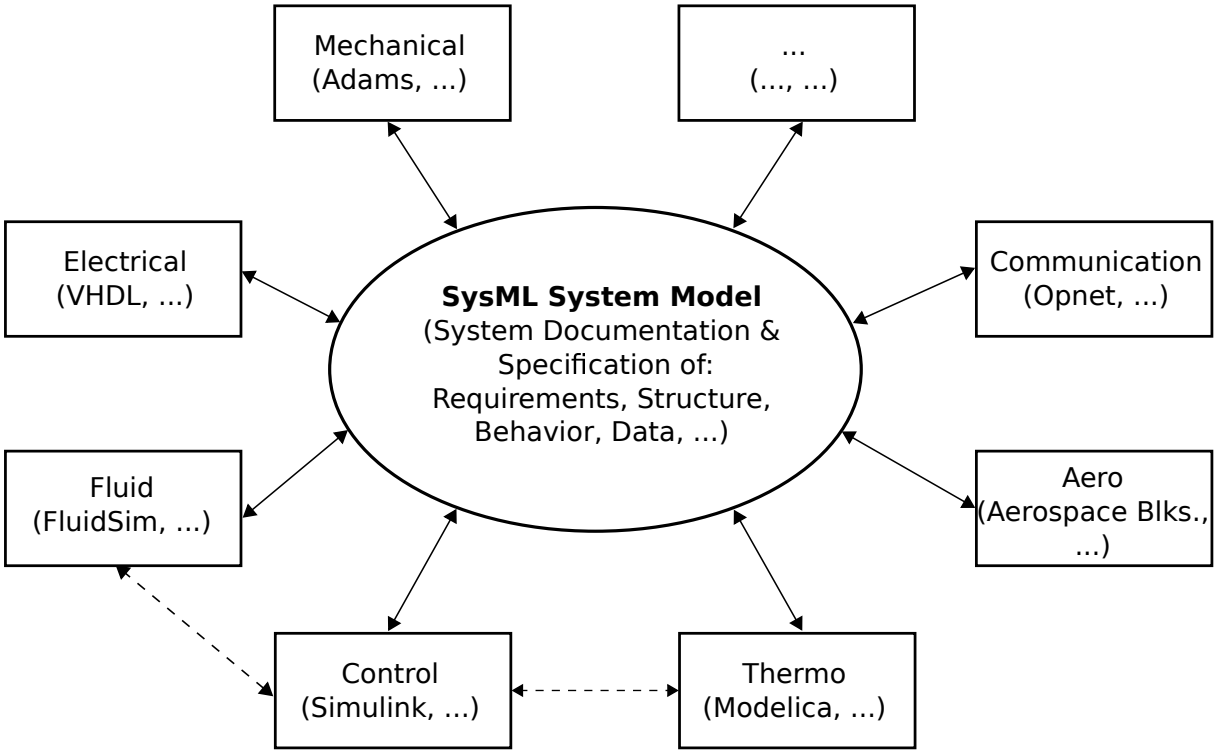


Figure 2.5: Interrelationships between M&S spaces in the engineering of complex technical systems. The M&S tools given are only selected examples to demonstrate the variety. M&S cross-relationships (dashed lines) between the technical spaces are only indicated by two examples. References for the mentioned softwares: Adams [69], VHDL [80], FluidSim [4], Simulink [87], Modelica [91], Aerospace Blks. [88], Opnet [104].

In addition to the model-driven development process, one objective of MDE is the reuse of models. Due to the cooperation of different technical spaces in the engineering of complex technical systems, the problem of designing an overall simulation quickly follows when using different tools. As a solution to avoid reimplementations, the FMI was developed as a general interface for simulation models. In the sense of the three model levels of the MDA, FMI defines a standard on PIM level. This makes it possible to generate different PSMs automatically on the basis of FMI. The problem of the automated transformation from the system model to the DSL is not solved by FMI. The fundamentals of FMI will be discussed in the next section.

2.4 The Functional Mock-up Interface as General M&S Interface

First, the problem of a general M&S interface is discussed. Then, in separate subsections, individual aspects of the FMI as a standard for a general M&S interface are discussed.

2.4.1 Problem Definition of a General Model Interface

Model-driven development of complex technical systems requires developing complex models from different domains, testing them individually and validating them as an overall model. Plateaux et al. [117] emphasize the importance of early testing of all design layers of a model. Van der Auweraer et al. [151] propagate the integration of different steps of a system design into a holistic industrial design process and thus the integration of models of subsystems from different domains. The implementation of the required holistic design process poses problems, which Gomes et al. [59] identify as follows.

- Simulation models are mostly developed in domain-specific M&S environments. Every environment has a specific user interface for creation of models and is optimized for the needs of a domain. The interface for saving and opening models is software specific as well. The interchangeability of models between different M&S tools is therefore not guaranteed.
- Intellectual property: System components are developed at great expense in terms of time and money. Therefore, the developer of a component often does not provide the entire model to the customer in readable form, but only the interface and encapsulated code.
- Model, software, hardware, and human-in-the-loop tests: In holistic development, even small modifications in subsystems and interactions with humans must be successively tested locally and in the overall context.

A general interface is necessary for M&S of cross-domain systems. To protect intellectual property, the interface should consider data encapsulation mechanisms so that the implementation of the model is not exposed. Furthermore, often the modeling power of a DSL is needed for an application, but when supporting a general model interface the simulation can also be executed with other simulators. Running a model with different simulators can be used to test *operational validity*. According to Sargent [132], operational validation comprises the simulation model and the executing simulator. Junghanns and Blochwitz [71] refer to the necessity to test simulation models with different simulators, such as using the DASSL (Petzold [116]) algorithm, with the argument: *"DASSL of tool A differs from DASSL of tool B, even if the same name is used. Solvers contain heuristics, which are optimized with respect to the models which are most commonly simulated."* Further arguments for testing operational validity through a general model interface are given in Pawletta and Folkerts [108]. It is not the core of this thesis to develop methods for testing the operational validity of simulators. However, the general model interface in combination with the methods to be developed in this thesis offer an approach to investigate the operational validity of simulators in practice.

So far, the problem of a general interface has been considered from the point of view of model exchange between simulators. Using model exchange, a cross-domain system can be modeled with different DSLs, the individual models can be integrated into an overall model, and this can be executed on a simulator. This approach corresponds to class 1 of the general classification of M&S approaches in Figure 1.1 in Chapter 1. Another approach follows from class 2 of the general classification, which is called co-simulation (Gomes et al. [59]). In a co-simulation, simulators running domain-specific models are coupled and coordinated via a master algorithm. The simulators and their models are considered in a co-simulation as a "black box" with a dynamic behavior, which provides an input and output interface.

For some domains there are de-facto interface standards. However, a general purpose standard for an M&S interface is desirable. For physically oriented technical systems, the FMI [92] has been defined as such a standard and is extensively used in practice. In the next subsections, FMI is discussed in more detail. First, the development of FMI is reviewed. Then FMI is discussed as a basis for model exchange and for co-simulation, although the method of co-simulation is not in the focus of this work. Finally, an outlook on a new FMI standard is given, which is interesting for further work.

2.4.2 Development and Overview of the Functional Mock-up Interface

The introduced challenges lead to the development of the DSblock interface by Otter (Otter [106], Otter and Elmqvist [107]). DSblock is the abbreviation for Dynamic System block. It is an open and general interface of models defined in the Fortran77 programming language. This interface is realized with input-output blocks, which represent the considered system. Thus a DSblock describes a general nonlinear dynamic system in a neutral way using ordinary differential equations or differential-algebraic equations. A DSblock can be parameterized and initial conditions can be set.

The latest definition of the DSblock model interface is version 4.0, which solves several challenges. The DSblock definition is related to the Modelica language, a hierarchical structure of DSblock models is supported, and the limitation to input-output blocks is cancelled. The target language of a DSblock is changed to C++. The version 4.0 of DSblock can be seen as predecessor of the FMI definition, which was released in 2010.

According to Blochwitz et al. [13] tool independent languages need to easily support model exchange between simulation tools. As previously discussed simulation tools are specialized for different domains and a complex system can be composed of a variety of model components. These components can be developed by different companies with special knowledge in a field. Thus the exchange of model components as well as the know-how protection is an essential task.

Next to proprietary interfaces of M&S tools FMI was developed as open and tool independent standard with the aim of support for model exchange between different simulation tools as well as co-simulation. The focus is on time-continuous and physical systems. The FMI standard is managed as Modelica Association Project and was continuously developed. In 2014 version 2.0 of the standard was released as discussed by Blochwitz et al. [14]. It combines the FMI for Model Exchange and the FMI for Co-Simulation and introduces

optional features. Very recently the FMI 3.0 standard is discussed (Junghanns et al. [72]), which extends the FMI 2.0 standard with additional functionality. In this work the FMI standard in the version 2.0 is used.

A model implementing the FMI is called *Functional Mock-up Unit* (FMU). An FMU is a zipped file with the file extension fmu. It comprises of the description schema in form of an *Extensible Markup Language* (XML) file *modelDescription.xml*, the C source code and/or static link libraries in binary form, and optional further data like documentation or icon. Static link libraries can be *Dynamic Link Library* (DLL) files or shared objects depending on the *Operating System* (OS). This is clarified in Figure 2.6. In case of FMI for co-simulation additionally the solver algorithm is part of the FMU.

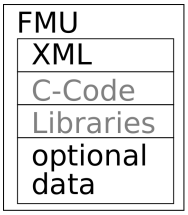


Figure 2.6: The structure of an FMU.

The FMI enables a simulator to generate executable simulation code from the FMU. Therefore there are a set of functions defined for FMI implemented in the target simulator. The model description stored in the XML file describes the interface of the FMU and contains the definition of all exposed variables of the FMU. The XML file can be read with functions of a simulation tool's programming language. The XML file specifies whether the FMU is defined for model exchange or for co-simulation and provides details on the model structure. The C-code maps the interface of an FMU and consists of a set of header files which define the C-types and C-interfaces. The FMU can be shipped with the C source code. In case the source code of an FMU is not integrated, the libraries are given in the FMU.

Starting in the automotive industry a number of tools from different fields support FMI [93]. There are domain-specific M&S tools as well as general-purpose M&S tools. Chen et al. pronounce in [31] the need for a generic implementation of FMI in Modelica-based M&S tools. The Modelica Association Project FMI provides tools to check FMUs and libraries for the implementation of the support for FMI in simulation tools, such as FMPy [27], the FMU Compliance Checker [95], the FMU *Software Development Kit* (SDK) [121], the FMI Library [94], or the FMI++ library by Widl et al. [159].

FMI is continuously further developed to this day. However, FMI is developed with the focus on time-continuous systems, whereas discrete time systems are neglected as pronounced by Widl et al. [160]. Franke et al. [54] describe the challenges and the progress of extending FMI for discrete time especially for control applications. As Müller and Widl in [97] and [98] point out, another challenge is the usage of FMI components in discrete event systems. For the M&S of complex CPS, FMI is of great interest despite the still existing limitations, since CPS contain a high number of time-continuous subsystems from different domains.

2.4.3 The Functional Mock-up Interface for Model Exchange

The intention of FMI for Model Exchange is the reuse of component models in different simulation tools. A component model is described by differential, algebraic, and discrete equations with time and state events. A M&S environment can generate C-code of a dynamic system model, which can be imported in other M&S tools. Due to the usage of C source code FMUs can also be used for simulation in embedded systems on microprocessors. When FMUs for model exchange are used, the FMUs are embedded in a target model and are simulated with a simulator of the target simulation environment. The models do not integrate any solvers. This is illustrated in Figure 2.7 and 2.8.

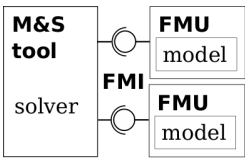


Figure 2.7: Relationship between FMUs for model exchange and the executing M&S environment.

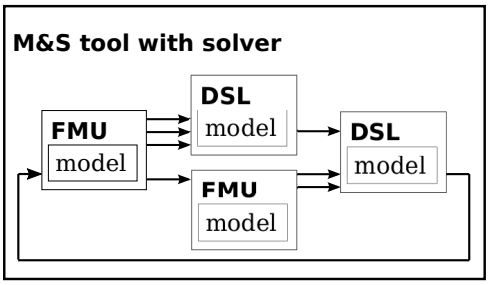


Figure 2.8: Structure of a simulation model with two domain-specific model components (DSL models) and two FMUs for model exchange in a target M&S environment.

2.4.4 The Functional Mock-up Interface for Co-Simulation

FMUs for co-simulation additionally contain the solution algorithms necessary to simulate the model. FMI for co-simulation is organized in a master-slave concept where subproblems are described by the slaves and the master organizes the simulation. The FMUs represent the slaves. An M&S tool needs to implement a master algorithm, which coordinates the simulation and the exchange of data between the FMUs. During simulation every FMU is solved by the embedded solver and data are exchanged between the FMUs at discrete communication points. Bastian et al. present concepts for a master algorithm in [8]. This general principle is illustrated in Figure 2.9.

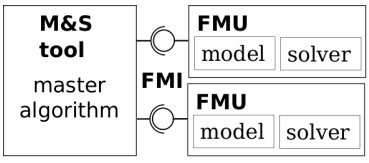


Figure 2.9: Relationship between FMUs for co-simulation and an M&S master tool.

2.4.5 System Structure and Parameterization Standard

A complex system is often modeled in a modular manner as a network structure as discussed in Section 2.2 and illustrated in Figure 2.8. Köhler points out in [78] that the FMI standard lacks:

- the possibility to separate the parameter data from FMUs,
- change parameters for an FMU independently from the simulation environment,
- the mapping of parameters in a network of FMUs,
- store a network of FMUs independent of the tool.

Therefore the new standard *System Structure & Parameterization* (SSP) as companion standard for the FMI standard was defined, which is managed as Modelica Association Project. The SSP standard defines a package to store FMUs, their connection structure, and their parameterizations in a zipped package independent of an M&S tool. Thus all system structures and all parameterizations are encoded in one package. An SSP package consists of the components as shown in Figure 2.10.

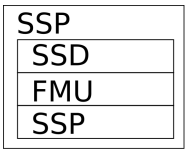


Figure 2.10: Components of an SSP package.

One part of an SSP is the *System Structure Definition* (SSD). In the SSD the connections of the FMUs with optional hierarchical subsystems in the SSP package are defined. Thus multiple SSDs are needed to specify several variants encoded in an SSP. The exchange of parameter data and their mapping to FMUs is included in the SSP. Since all variants – the structures and the parameter configurations – are pre-defined in the SSP, the SSP can be classified as 150% model approach. The 150% model approach (Alt [2]) is discussed in detail in Section 2.5.2. The SSP companion standard was adopted and released at the Modelica Conference in March 2019 and was only recently included in non-commercial simulation tools. Therefore the SSP standard is not considered in this work.

2.5 Variant Modeling, Selection, and Model Generation

Variant modeling and selection are broad terms in software engineering. Model engineering of a FoM for simulation is closely related to the modeling of system variability in software engineering with respect to variant modeling and selection. System variability is defined as the ability of a system to be configurable, extendable, or adaptable depending on its purpose and objective. Model generation means in this context to generate executable simulation models for selected model variants. Schmidt [135] summarizes the steps 2 to 6 of the life cycle of a FoM for simulation in Figure 2.1 under the term *variant management*. Figure 2.1 also shows that there is a close correlation between variant management and the

simulation of model variants and their evaluation. As already motivated in Chapter 1, the investigation of complex systems requires a (semi)automated derivation of model variants and model generation, since the enormous effort to create and test individual variants manually is hardly manageable. According to Schaefer et al. [134], a central variant specification and automated variant selection and evaluation reduces the maintenance effort of complex models, since modifications are specified centrally and are automatically included in model selection and model generation.

In this section, established approaches for modeling model variants, the selection of candidate models, and the generation of executable simulation models are discussed. For a pragmatic explanation of selected aspects, a minimal case study is introduced first, which will be referred to again in later chapters.

2.5.1 Introduction of a Minimal Case Study of a Family of Models

The case study presented in this section is used throughout this work to illustrate the methods which are presented and introduced. The case study describes a feedback control system with optional feedforward control. It is modeled in a signal flow-oriented manner using transfer functions to describe the behavior of the components in frequency domain. Controlled variables in a feedback control system are usually influenced by disturbances. A common approach for minimizing the influence of predictable disturbances is adding a feedforward control.

In the feedback control system a PID controller controls a process unit with a *PT1* behavior. A disturbance with a *PT1* behavior affects the output of the process unit. The FoM described in this case study includes two structure variants, either with a feedforward control or without a feedforward control. For every structure variant a range of different parameters can be applied to the PID controller. Figure 2.11 depicts a schematic representation of the application.

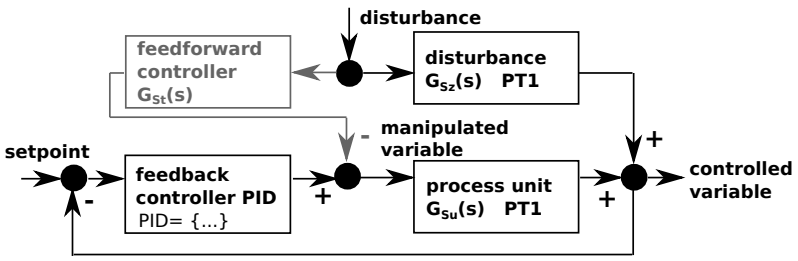


Figure 2.11: Structure of the feedback control system with optional feedforward control.

The system's behavior follows the *PT1* transfer function in Equation 2.1 and the step-shaped disturbance affects the output of the process unit with a *PT1* behavior according to Equation 2.2. The optional feedforward control is realized by subtracting the disturbing signal calculated by Equation 2.3 from the manipulated variable.

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1} \tag{2.1}$$

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1} \tag{2.2}$$

$$G_{St}(s) = \frac{G_{Sz}(s)}{G_{Su}(s)} = \frac{20 \cdot s + 1}{10 \cdot s + 1} \tag{2.3}$$

The control objectives are to ensure that overshoot does not exceed a certain value and to maintain a certain settling time. The goal is to find the best control structure that reaches the control goals. In this example, the best control structure means the most minimal structure possible with controller parameters that achieve the control objective.

2.5.2 Approaches from Software Engineering

Based on the definition of a FoM given in Section 2.1, a FoM has many relations to a *Software Product Line* (SPL). SPL is the mostly used approach in software engineering for modeling the variability of a software system. The methods and tools used to develop SPLs are referred to as *Software Product Line Engineering* (SPLE). In an SPL the commonalities and differences between software products are specified (Clements and Northrop [34]). The aim is to guide the structure, reuse, and describe variations across all phases of the software lifecycle (Apel et al. [3]), to reduce time to market of products, and to increase product quality (Schaefer et al. [134]). Each feature of a product is a characteristic or end-user-visible behavior of a software system. Thus, the modeling of variability is necessary.

Section 2.2 discussed different levels of model abstraction when introducing MDE and MDA. Pawletta et al. [114] as well as Jafer et al. [70] state that variant or variability modeling can take place at different abstraction levels. For this purpose, they refer to the *Meta-Object Facility* (MOF) hierarchy [100], which has strong relations to the model classes defined by the MDA. The MOF hierarchy distinguishes four levels of abstraction, called M0 to M3. M0 describes a system in reality and M1 a model of the system in a modeling language, while the abstraction level M2 defines a metamodel of the modeling language. The abstraction level M3 is the MOF. Approaches described next are categorized according to this hierarchy.

Feature Modeling and 150% Modeling A widely used modeling approach is Feature Modeling as described in Kang et al. [73], Deursen and Klint [43], or Apel et al. [3]. *Feature Model*s (FMs) were introduced as part of the Feature-Oriented Domain Analysis and are usually developed by experts of a domain. The developers need to know which features are combinable and which features need the presence of other features. So the validity of feature selections must be defined. Although this can be done using a list, in linguistic form, or using modeling formalisms such as UML, a graphical representation using *feature diagrams* has proved to be the most clearly arranged way (Apel et al. [3]). The semantics are specified by translation into propositional logic. The feature diagrams then define the FM as a hierarchy of features and constraints. The relations between features in FMs can be *mandatory*, *optional*, *alternative*, or *or*. Additional constraints can be defined with cross-tree constraints.

Figure 2.12 shows an example of the specification of the variants of the simple control system introduced in Section 2.5.1 in the form of an FM. For the PID controller alternatively

a P controller can be chosen. Both can be combined with the two structural variants discussed. The system is divided into a *basic feedback control system*, a *disturbance*, and an optional *feedforward* control. The basic feedback control system consists of the mandatory components *setpoint*, *feedback*, *feedback controller*, and the transfer function representing a *process unit*. In this case study a *step* shaped disturbance transformed by a *transfer function* is described. For adding this disturbance to the basic feedback control system an *add* feature is needed. The optional feature feedforward control needs a *transfer function* and a *subtract* feature.

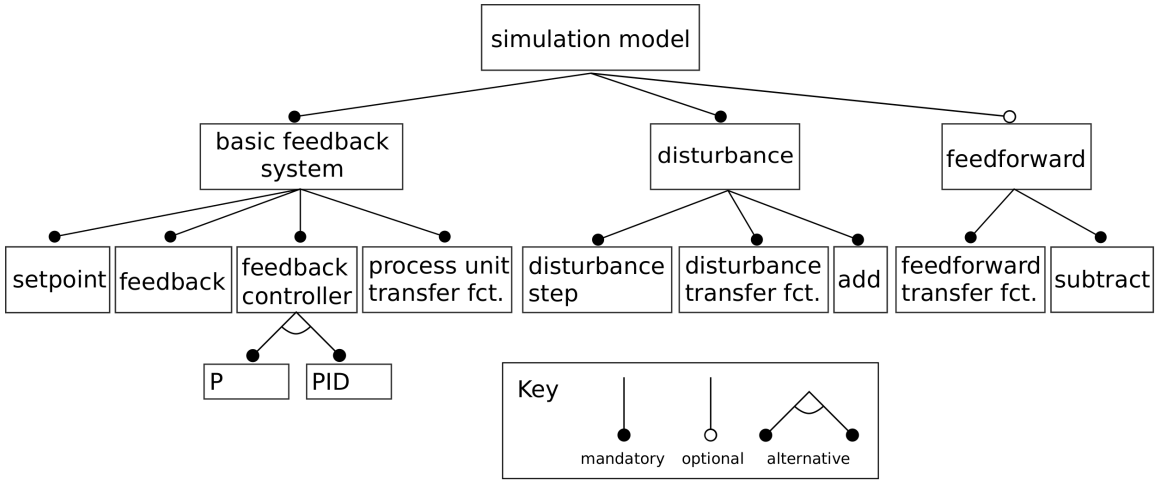


Figure 2.12: FM for specification of variants of a simple control system.

As shown in the example in Figure 2.12, FMs can be used very well to specify compositions of components on network level and parameterization variants of components. Coupling relations between components and dynamic system behavior cannot be described with FMs. Thus, FMs are only conditionally suitable for the specification of the design of a FoM according to Step 3 in Figure 2.1. The modeling with FMs corresponds according to the MOF hierarchy to the abstraction level M2.

In M&S practice, FMs are often combined with the 150% modeling approach (Alt [2], Weißleder and Lackner [157]). As mentioned before, a 150% model is specified with a DSL and represents all model variants of a FoM at M1 level. Variation points are introduced to model the choices. The variation points have a parameter over which the selection can be controlled (Grönniger et al. [62], Kolassa et al. [79], Gutekunst and Weiland [63]). The parameters of the variation points are used to define links between the FM and the 150% model. Figure 2.13 shows a 150% model of the feedback control case study, specified with a DSL as a signal flow graph. Here, the blocks *procUnitSys*, *tfDist*, and *tfFeedforward* represent the transfer functions mentioned in Section 2.5.1, and *ctrlPIDSys* represents the PID controller. Components of the basic control system mentioned in Figure 2.12 are highlighted in red in Figure 2.13, components of the disturbance in yellow and the components of the feedforward in green. The switch block with gray background defines a variation point between model structures in the 150% model.

The FM is used to select one specific system variant. The selection is done by setting the parameters of the variation points of the 150% model in the FM. This approach is classified as a *subtractive* way of model generation, because parts are taken away from the maximum overall 150% model. Transformation methods in the sense of MDA are

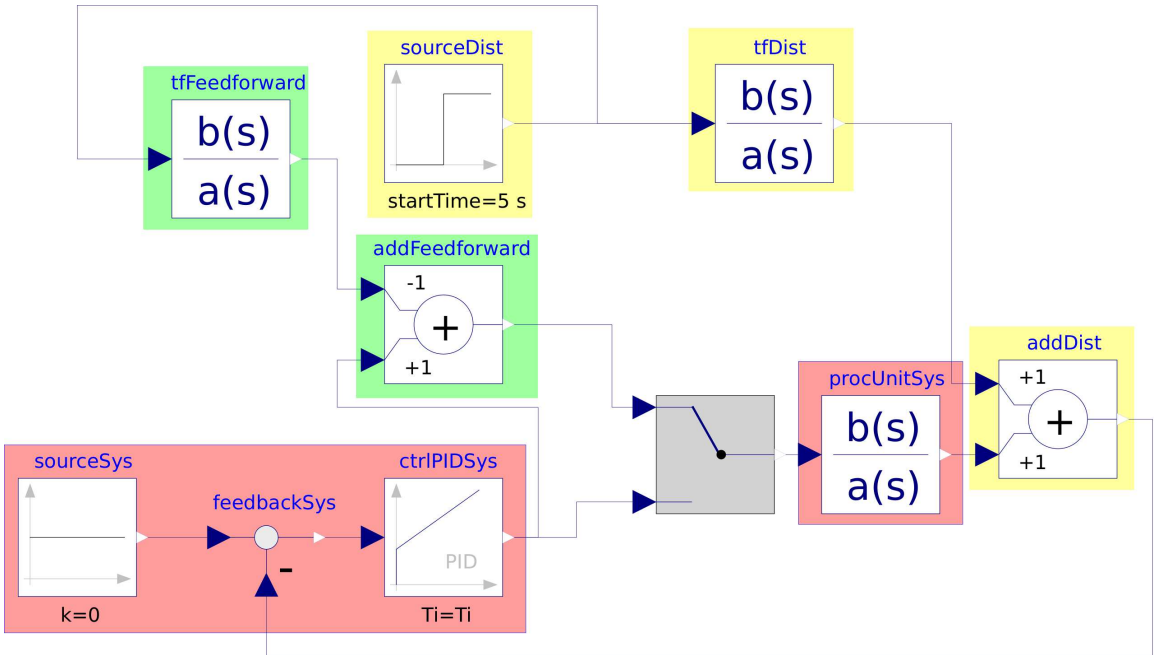


Figure 2.13: 150% model of the introduced control system.

not required, because only a PIM to PSM translation is performed in the domain-specific M&S environment.

An implementation of FMs in the context of software engineering is the software pure::variants [120]. This software can be used in combination with a simulation tool such as MATLAB/Simulink, OpenModelica, or Dymola. MATLAB/Simulink also provides an integrated variant manager, which follows the selection from a 150% model approach [86].

Other Approaches An extension of feature-oriented programming is the *delta* modeling approach like in Schaefer [133] or Clarke et al. [33]. A core product and deltas of this core product of a product line are identified, where the deltas specify modifications of the core product. Rules for these deltas specify the need of modifications for a feature configuration. This approach can be used for automated product derivation in SPLE as discussed in Schaefer et al. in [134]. There are several approaches how to gain a consistent model.

Müller et al. uses a SysML-based approach in [96] to describe the variants of a system at M2 level. Thereby SysML-based components reference simulation component models in a simulation library.

Furthermore, approaches exist to extend SysML by dynamic descriptive language elements (Reichwein et al. [123]), so that SysML can be used to specify model structures, model parameterizations as well as dynamic behavior.

Pros and Cons A disadvantage of feature-oriented modeling is the lack of concepts for specifying coupling relationships between components, which means that system structures in the sense of M&S cannot be modeled completely. The connection of FMs with 150% models or delta models becomes unclear with complex models and complicates the model maintenance. The workload for creating and testing an overall model with all model variants according to the 150% approach corresponds to the workload of testing with

single models. The advantage is the centralized incorporation of model changes and the elimination of transformation methods in the sense of MDE/MDA for generating an executable simulation model. Thus, the approach can be used quickly in practice. However, approaches for solving the central problem of M&S of complex technical systems, to generate an overall simulation model on the basis of partial models of different DSLs, are not known.

Linking SysML-based components with component models in a simulation library is a good approach from the M&S point of view. The system description is clearly separated from the behavior, which is coded in the simulation components in a simulation library. The components are coded in a specialized DSL. A transformation method then generates a simulation model. Thus, principles of MDE and MDA are supported. It should be emphasized that in the approach the mapping from SysML components to components in the simulation library must be done depending on the simulation library used.

The extension of SysML by dynamic descriptive language elements seems obvious from the software engineering point of view and supports the principles of MDE and MDA. However, this approach contradicts previous experiences in the technically oriented M&S domain, which is characterized by specialized DSLs and simulation tools of the various technical spaces and conditioned the development of FMI according to Section 2.4.

2.5.3 Classic System Entity Structure and Model Base Approach

Analogous to approaches coming from software engineering the systems theory community introduced methods for platform-independent variability modeling and model selection with subsequent platform-dependent model generation. Zeigler [163] developed with the *System Entity Structure* (SES) one of the first high level approaches for variability modeling in the context of M&S problems. The SES is a structural knowledge representation scheme to specify decomposition, taxonomy, and coupling relations of a multifaceted system.

Basics of the SES/MB Approach According to the definition of a FoM, an SES describes a set of modular, hierarchical system structures. From the point of view of data structures, an SES is a tree structure. The tree is composed of four types of nodes interconnected by edges. One type of nodes are entity nodes. Entities describe real or imaginary objects. The other three node types describe relations between entities and are referred to collectively as descriptive nodes. In the context of M&S leaf *entity nodes* can be linked to basic models organized in a library called *Model Base* (MB). Moreover, entity nodes can specify parameter settings for referenced basic models. A basic model is an encapsulated dynamic system with a defined input and output interface. A basic system can be any atomic or coupled model regarding to the dynamic system specifications in Section 2.2, if the basic model satisfies the closure under coupling property as defined in Zeigler et al. [167].

According to Rozenblit and Zeigler [128], Zeigler et al. [167], or Zeigler and Hammonds [164], an SES can define coupling relations between entities and specify rules by which an SES can be pruned. A pruned SES that describes a concrete system structure including parameterization of referenced basic systems is called a *Pruned Entity Structure* (PES). The pruning of an SES is in the sense of MDE/MDA a transformation method, which is

called *pruning*. The pruning method is used to select model variants.

The SES/MB framework according to Zeigler et al. [167] describes the integration of an SES with an MB. The SES/MB framework defines another transformation method, which is called *build* here following Pawletta et al. [114]. With the build method, a complete simulation model is generated using the PES and basic models from the MB. The resulting model is an optimal taylored model in contrast to 150% models. In the terminology of variability modeling this kind of model generation is an *additive* approach. Figure 2.14 shows the approach to M&S based on the classic SES/MB framework.

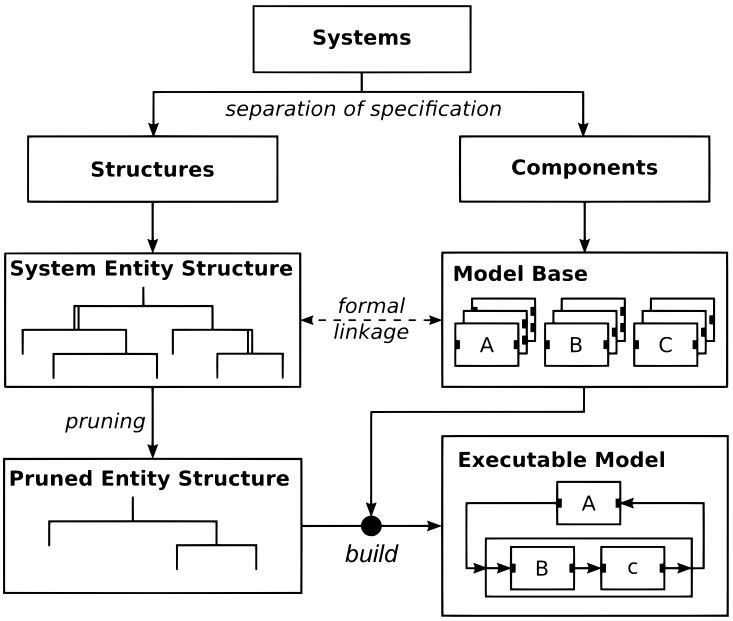


Figure 2.14: Procedure model and SES/MB framework modified based on Zeigler et al. [167].

Based on the classic SES/MB framework, various M&S tools have been implemented. Examples are the Java-based MS4Me environment by Zeigler and Sarjoughian [165] or the MATLAB/Simulink-based prototypes by Hagendorf and Pawletta [65] and Schmidt [135]. The implementations are based on simulator-independent SES models and simulator-specific MBs for model generation. Consequently, the generated executable models are simulator-specific PSMs.

The SES/MB Approach in Context of MDE/MDA According to Zeigler and Hammonds [164], the SES is an ontology, a language with syntax and semantics to represent declarative knowledge. It is suitable for describing system configurations for different application domains. As Zeigler and Hammonds describe in [164] the development of the UML in the field of software engineering influenced the developers of ontologies and vice versa. A language to represent ontologies is the *Web Ontology Language* (OWL). According to the *World Wide Web Consortium* (W3C) [155] it is designed to represent knowledge about things, group of things, and relations between things. Zeigler and Hammonds compare the OWL with the SES in [164]. Like the OWL, the UML is primarily used for platform-independent specifications in software engineering. The main application area of SES is the application-independent specification of modular-hierarchical system configurations. Thus the SES corresponds to the UML and OWL. In analogy to UML and OWL, the SES ontology is assigned to the M3 level of the MOF hierarchy.

In the context of this work, an SES can be used to describe all possible model configurations of a FoM with respect to the different model structures and component parameterizations. Zeigler and Hammonds describe in [164] the relationship between the representation of systems as UML diagram and an SES. The most obvious difference is the representation of the SES as tree structure, where UML uses general graph structures to express objects and their relations. Durak et al. propagate in [46] the SES/MB framework as an MDE approach in the technical systems simulation domain.

According to Gorton [61] any language can be made MDA compatible: *"The MOF also provides mechanisms to determine how any model defined in a modeling language can be serialized into XML documents or be represented by programmable interfaces. Any existing modeling language can be made MDA compatible by creating a MOF representation of the language."* An XML representation is defined for the SES, such as described by Zeigler and Hammonds [164] and by Zeigler and Sarjoughian [165], thus fulfilling the key requirement of interchangeability of models according to the MDA.

The transformation method *pruning* is used to select a concrete model configuration from an SES in the form of a PES. A PES specifies the selected variant platform-independently in the SES language. From the point of view of the MOF hierarchy, a PES represents a model of the M1 level.

With the transformation method *build*, a simulation model is generated on the basis of a PES for a selected model configuration using basic models of an MB. The generated model is a complete simulation model which represents the model structure, the model dynamics, and the model parameterization. Depending on the basic models, the generated simulation model is according to the MDA model classification:

- a PSM based on a general programming language or a simulator-specific DSL or
- a PIM based on a formal or simulator-independent specification, which is transformed into a PSM by transformation methods of the respective M&S environment used.

As discussed in the previous paragraph, previous implementations of the SES/MB framework use only simulator-specific MBs.

Pros and Cons The SES/MB approach supports a platform-independent specification of model configurations with respect to the possible model structures and component parameterizations. The modeling of the dynamic behavior takes place in the form of basic models, which are organized in an MB. Two transformation methods are used to generate customized models of selected model configurations. All previous implementations of the SES/MB framework are based on simulator-specific MBs. The primary requirement for the engineering of complex systems, such as CPS, to generate complex overall models using submodels of different DSLs, cannot be implemented in this way.

2.6 Summary

Starting from the definition of a FoM, a life cycle model for FoMs was derived based on generally known life cycle models of M&S. Subsequently, basics of formal modeling of dynamic system behavior and the fundamental relation between model and simulator

were considered. No FoM specific requirements followed from the consideration. The subsequent analysis of basic principles of MDE and the associated MDA of software engineering in the context of M&S of complex technical systems revealed special features of system development. This follows from the multiplicity and technical differentiation of the technical spaces involved. In the context of the requirement specification, the system and data analysis, and partly with the conceptual modeling, one works on basis of a common system model as in software engineering. Documentation and development results also flow into the central system model. The detailed development in the individual technical spaces, however, is largely carried out using different methods and software tools. With respect to the M&S domain, different simulation systems with specific DSLs are used. The classical implementation of the MDE/MDA principles would require an enormous amount of transformation methods.

For the realization of domain-spanning overall simulation models on the basis of submodels of different technical spaces, the M&S community developed:

- multiphysics modeling languages like Modelica as a replacement for specific DSLs and
- general model interfaces, such as FMI, for the exchange of models between different simulators or for the coupling of different simulators (co-simulation).

Finally, FoM-specific aspects such as the modeling of different model configurations and the most automatic possible derivation of candidate models with subsequent generation of executable simulation models were considered. Here it became apparent that the methods adapted from software engineering are hardly suitable for solving the challenges of M&S of complex technical systems.

Furthermore, the classical SES/MB framework was considered. This is an approach that has already been widely discussed in the M&S domain in the context of FoMs and the principles of MDE. While the SES concept fulfills the basic requirements of MDE, such as platform independence and consistency from conceptual modeling to model generation of executable simulation models, deficiencies were identified in the MB concept. All previous approaches and implementations of the SES/MB framework are based on simulator-specific MBs and thus on simulator-specific transformation methods for model generation. In order to generate complex overall models using submodels of different DSLs, the submodels must support a general interface and be organized as basic models in an MB. This concept will be investigated in this thesis using FMI for model exchange and an extended SES/MB architecture. Besides the conception of a general transformation method for the generation of FMI-based simulation models for different simulators, simulator-specific considerations for model generation as well as for the automation of goal-oriented simulation studies for FoMs will be made.

3 Extensions to SES/MB-based Modeling and Linking With Simulation Experiments

Starting from the basics of the previously introduced classic *System Entity Structure* (SES)/*Model Base* (MB) framework as a high level approach for *Modeling and Simulation* (M&S) of a *Family of Models* (FoM), this chapter starts with the consideration of further details of the classic framework. Then existing and newly developed extensions to SES/MB-based modeling are discussed. The considered extensions refer to modeling with the SES and operations for processing of an SES with the goal of an automated selection of candidate models of a FoM. Aspects of SES/MB-based modeling in the context of simulation-based experiments are then discussed. The concept of the experimental frame as an interface of a model to simulation experiments and the structural design of simulation-based experiments are discussed.

3.1 The Classic SES/MB Approach in More Detail

In Section 2.5.3 the SES/MB framework with its fundamental components and operations was discussed. An overview regarding the SES/MB framework was already given in the introduction in Figure 1.2. In the following, based on Zeigler et al. [167] and Zeigler and Hammonds [164], some basic aspects are discussed in more detail.

The SES was introduced in Section 2.5.3 as a tree structure consisting of entity nodes and descriptive nodes. In the context of M&S, entity nodes represent atomic or coupled systems, while descriptive nodes describe relationships between entities. Accordingly, the root node and leaf nodes are always entity nodes. The referencing of models in an MB by entity nodes has already been discussed in Section 2.5.3. Furthermore, entity nodes can define attributes, called *attached variables*, to specify properties of models, such as parameter settings. At this point, the three types of descriptive nodes and the axioms for the construction of an SES shall be considered in more detail.

Aspect nodes describe how entity nodes can be decomposed into partial entities, whereas the taxonomy of an entity is described by *specialization nodes*. Thus, aspect nodes describe a *has-a* relationship, whereas specialization nodes describe an *is-a* relationship. Like aspect nodes *multi-aspect nodes* describe the decomposition of an entity, but with the special property that all child nodes are of the same type of entity. Multi-aspect nodes have an attribute, called *Number of Replications* (NumRep), that specifies the number of children to be created when pruning an SES. Moreover, aspect nodes and multi-aspect nodes can

define relationships between the parent of the node and the children in an attribute. A *coupling relation* in the sense of modular-hierarchical model structures is defined by two pairs, each consisting of an entity and a port name, such as *(source entity, source port, sink entity, sink port)*. No special attribute is defined for specializations in the classic framework. However, *selection constraints* can be specified. This is done with a special edge type between selection-dependent specializations.

In order to build the tree structure semantically correctly, *axioms* for the SES are defined. The types of the nodes in a tree path have to follow the axiom *alternating mode*. Every entity node has to be followed by a descriptive node and vice versa. A *strict hierarchy* is needed. In every path of the tree, a name of a node is allowed to occur only once. If nodes in different paths have the same name, they need to have the same attributes and isomorphic partial trees. This is called *uniformity* and it implies that the partial tree below nodes with the same name in an SES does not have to be repeated. Nodes with the same father, called sibling nodes, have to be *valid brothers*, meaning that sibling nodes must not have the same name. The axiom of *attached variables* implies that a node must not have variables (attributes) of the same name. The axiom of *inheritance* implies, that when pruning an SES, the parent and the child of a specialization node combine their attributes. If parent and child have the same attributes, the parent's attributes are overwritten with the child's attributes and their values.

With the three types of descriptive nodes and the selection constraints or semantic conditions, different model variants can be described in different ways with respect to the model structure and the model parameterization. In terms of variability modeling, descriptive nodes of the type specialization and multi-aspect as well as siblings of aspect nodes define variation points. The selection of a model variant is done with the pruning operation, as introduced in Section 2.5.3 in principle. Zeigler and Sarjoughian discuss in [165] different pruning approaches. The pruning can be done interactively or automated. When doing the pruning interactively the user needs to have the knowledge on how to find decisions at the variation points. Pruning an SES interactively means to manually decide at each variation point to find a certain system configuration. This is a time intensive and error-prone process which is hard to carry out (Zeigler and Sarjoughian [165]). Thus, the automation of the pruning process is desirable. One approach for automated pruning is the enumerative pruning as described by Zeigler and Sarjoughian in [165]. When following the approach of enumerative pruning all *Pruned Entity Structure*s (PES) coded in the SES are produced once. Due to the high number of variants in engineering problems the interactive pruning is often not feasible, while enumerative pruning can be neglected since this approach is not goal-directed. A goal-directed automation of the pruning and model building process requires an extension of the SES/MB approach. Knowledge on how to find decisions to resolve the variation points during pruning needs to be defined before starting the pruning process. In [165] the suggestion is made by Zeigler and Sarjoughian to specify rules for pruning in a script. Another approach is the extension of the SES with features to specify all knowledge necessary for automatic pruning in the SES as suggested by Pawletta et al. in [111]. The extended SES approach and an algorithm for the pruning operation are described in detail in the next sections. Regardless of the pruning approach, the result of pruning is a PES or a set of PES, where each PES describes the structure and parameterization of a concrete candidate model of a FoM.

The SES/MB framework combines the concept of SES with an MB, such as described

by Rozenblit and Huang [126], Rozenblit and Zeigler [128], or by Zeigler et al. [166]. The transformation method *build* already explained in Section 2.5.3 generates an executable simulation model from the information of a PES using basic models of the MB. The reference to the basic models in the MB is made via information coded in the leaf nodes of the PES. Furthermore, the leaf nodes can define concrete parameter settings of the basic models, which are determined when pruning the SES. Similarly, when pruning the SES, coupling information from aspect and multi-aspect nodes is transferred to the PES. Finally, it should be noted again that the SES and PES are simulator-independent, while previous implementations of the MB are simulator-specific.

3.2 Extensions of the Classic SES/MB Approach

The SES modeling approach has been continuously developed since its introduction by Zeigler in the 1980s (Zeigler et al. [167], Pawletta et al. [111], Pawletta et al. [110], Zeigler and Hammonds [164], Zeigler and Sarjoughian [165], Santucci et al. [131], Schmidt and Pawletta [136], Schwatinski and Pawletta [139], Pawletta et al. [109]). Schmidt introduced in [135] an architecture for automated simulation-based experiments based on the SES/MB framework. With the architecture Schmidt [135] developed a method of an automated, goal-directed selection of candidate models from an SES and a method for the generation of simulation models based on it. The model generation method is simulator-specific and the specification of the SES is also partially simulator-dependent. The objective of this thesis is to develop a largely simulator-independent SES/MB architecture, building on Schmidt's [135] architecture. Accordingly, some extensions to the SES/MB approach are discussed in the following subsections. It is started with extensions developed by the *Computational Engineering and Automation* (CEA) research group at the Hochschule Wismar, University of Applied Sciences, in the context of Schmidt's [135] architecture. Then, SES/MB-based operations are discussed in the context of the extensions under consideration. The focus here is on an automated pruning operation, which was newly developed by the author to remove limitations that exist in Schmidt [135]. Subsequently, essential concepts of the extended SES/MB-based modeling are demonstrated by means of examples. The first example refers to already known extensions, which are built upon in this thesis. The second example deals with new concepts, which have been developed by the author in cooperation with colleagues.

3.2.1 Selection Rules, Variables, Functions, and More

For automatic processing of an SES during pruning every decision for a selection at a variation point must be taken based on pre-configured rules. Theses rules shall allow decisions to be evaluated automatically during pruning. In the SES a selection rule can be specified as node attribute by

- (i) a specialization node or
- (ii) siblings of aspect or multi-aspect nodes.

In case (i) the rule is called *specrule*. A specrule defines the conditions for selecting a particular child node of the specialization when pruning the SES. In case (ii) the rule is

called *aspectrule*. An aspectrule defines the conditions for selecting a sibling from aspect or multi-aspect siblings when pruning the SES.

A rule can be based on variables called *SES variable*s (SESvars), which have a global scope for the SES. Thus, a rule can be defined conditionally based on the values of SESvars. The value range of these SESvars can be limited as well as dependencies between SESvars can be set by a *semantic condition*. Using semantic conditions the variant diversity coded in the SES can be limited.

Another extension are *SES function*s (SESfcns), which are a global attribute of an SES like the SESvars. SESfcns enable the specification of procedural knowledge in the SES. Some kinds of variability can be described more easily with SESfcns as with SES nodes. Typical examples include the definition of varying coupling relations or the definition of variable parameter configurations, as shown by Pawletta et al. in [111].

During pruning, the SES tree is reduced. Usually the tree is traversed several times. When resolving descriptive nodes, in some cases two or more aspect or multi-aspect nodes on the same level (new siblings) are created in the meantime, for which no aspectrule is defined. For automatic pruning it is necessary to sequence such nodes by giving a *priority*. The priority is a global attribute of aspect and multi-aspect nodes.

In the classic SES/MB approach, the models in the MB are directly referenced by the names of leaf nodes. This leads to a direct binding of node names of the SES and basic models of the MB. To achieve a decoupling of node names of the SES and basic models of the MB, a special attribute with the name *mb* was introduced. This attribute is defined for leaf nodes and gets as value the name of the model to be referenced in an MB. This value can also be assigned variably via an SESfcn depending on an SESvar. Further attributes for configuring parameters of a basic model can be defined for leaf nodes. All node-specific data – rules as well as attached variables – is referred to as attributes and attributes can be defined conditionally using SESvars and SESfcns.

Couplings as introduced in the classic SES/MB framework are extended for usage in combination with basic models of various type. Basic models can have different types of ports depending on their dynamic behavior and their purpose. In a model, components describing a signal flow-oriented system have a different type of coupling than components describing a physical system or discrete event system. Therefore, the specification of couplings is extended with fields for specifying the type of input and output port, such as *(source entity, source port, source port type, sink entity, sink port, sink port type)*. Usually the types of the source port and the sink port are identical. An example for couplings specified using an SESfcn are depicted in Figure 3.1. The port type is indicated by the word *type*.

Finally, a special element of the classic SES/MB framework, the *NONE* node, should be discussed at this point. This node can be used as an entity node under a specialization and always forms a leaf node of the SES. If the node NONE is selected when pruning an SES at a specialization, it means that no entity of this specialization will be integrated into the candidate model. Optional SES tree sections as well as the logical OR selection are expressed with the help of the NONE node. Thus, analogous to feature models, choices can be described according to the logical AND, OR, and XOR. Deatcu et al. discuss in [40] how to express the logical AND, OR, and XOR as well as options between nodes with the elements of an SES.

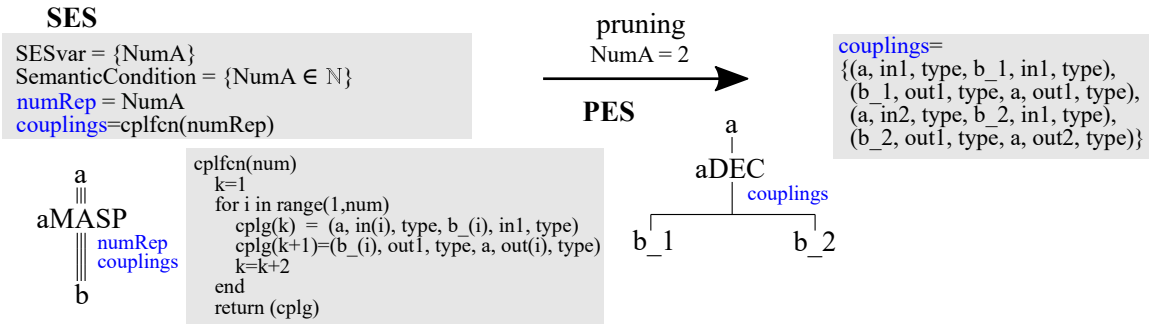


Figure 3.1: Couplings specified using an SESfcn at a multi-aspect node.

For the interchangeability of SES models according to the *Model-Driven Architecture* (MDA), an *Extensible Markup Language* (XML) notation has been defined by the author taking into account all extensions discussed so far as shown in Appendix D on an example.

3.2.2 Methods of the Framework

The classic SES/MB framework defines the two methods *pruning* and *build* already discussed. Furthermore, in the context of the classic framework, the operations *merge* (Zeigler and Hammonds [164], Zeigler et al. [168]) and *flattening* (Zeigler and Hammonds [164], Zeigler and Sarjoughian [165]) were introduced. All four operations were implemented as methods for the architecture to be developed in this thesis. In the following, the four methods are briefly discussed. The focus is put on the pruning operation, which has been comprehensively developed methodologically. A new development, the pruning of SES with hierarchical multi-aspects, is discussed separately in Section 3.2.4. The build operation, which was also comprehensively redeveloped, is explained here only in principle. A detailed description of the build operation is given in Chapter 4.

Merge On the SES, a merge operation is defined allowing two or more SES to be combined. In merging, the root node of an SES is united with a leaf node of a target SES. This allows the quick reuse of a once defined SES and supports a modular problem specification on the SES level. After merging, the target SES must be checked for compliance with the axioms in Section 3.1. The merge operation is performed in the SES modeling phase and has no influence on the automated derivation of candidate models. For the objectives of this thesis, no extended requirements for the merge operation follow.

Pruning The pruning operation is used to select one or more candidate models from a FoM. As explained in Section 3.1, there are different approaches for the pruning operation. The focus of this work is the automated pruning of an SES using the extensions discussed in Section 3.2.1. A first algorithm of an automated pruning operation was already developed by Schmidt in [135]. However, in Schmidt's algorithm the hierarchy depth of multi-aspect nodes is limited to one and the formation of sibling forms is limited to aspect siblings. The pruning operation developed in this thesis fixes these limitations.

In Section 3.2.1, extensions of SES regarding siblings of aspect or multi-aspect nodes have already been discussed. Zeigler and Hammonds [164] outline further sibling forms, which can be used for modeling FoMs. However, in some cases no clear rules for pruning are defined, as they are required for an automated pruning operation. With the collaboration

of the author, several design patterns for variability modeling using the SES and unique pruning rules for it have been defined by Deatcu et al. [40]. Appendix B briefly summarizes the patterns and their basic pruning steps.

In the following, node-type related essential steps of the new pruning operation are explained, which are summarized in the activity diagram in Figure 3.2. The diagram contains the essential operations for pruning the different forms of sibling nodes, according to the patterns shown in Appendix B, except for hierarchically arranged multi-aspect nodes. The unsolved problem of automated pruning of hierarchical multi-aspects is discussed separately in Section 3.2.4. The traversal algorithm for determining the next node is not presented in detail. Essential for the automated pruning is the coding of the targets for the selection of candidate models. The targets are coded by value assignments of the SESvars. The SESvars form the input interface for pruning the SES.

Starting the pruning operation at first the allowance to prune is checked. Compliance with the axioms of the SES according to Section 3.1 must be checked as well as further constraints, such as that the root and all leaf nodes in the SES are entity nodes. All SESvars are checked to fulfill their semantic conditions. Next, all SESfcn calls are evaluated with the current value assignments of the SESvars and all variable value assignments of node attributes are completed. After this step, all node attributes are set. Accordingly, the individual nodes of the tree can now be traversed and evaluated. In the following, the node type specific evaluations are briefly discussed.

Entity nodes
No special evaluation needs to be taken for entity nodes. They are transferred to the PES according to their specification in the SES. But, in connection with specialization nodes and multi-aspect nodes, renaming or attribute unification can be done.

Specialization nodes
Pruning a specialization node is linked to the axiom of inheritance. In a specialization node, a child of the specialization node is selected by evaluating the specrule attribute or a selection constraint relationship. After that, the selected child node and the parent node, which are both of the type entity, are unified. That is, their names and attributes are combined. In case of attributes with the same name, the attribute value of the child node is set. Furthermore, the subtree of the selected child node is attached to the newly created combined node. The original specialization node is deleted. Sibling nodes of type specialization are evaluated one after the other without sequence relationship. Specialization nodes in a sibling relationship with aspect or multi-aspect nodes are evaluated before them.

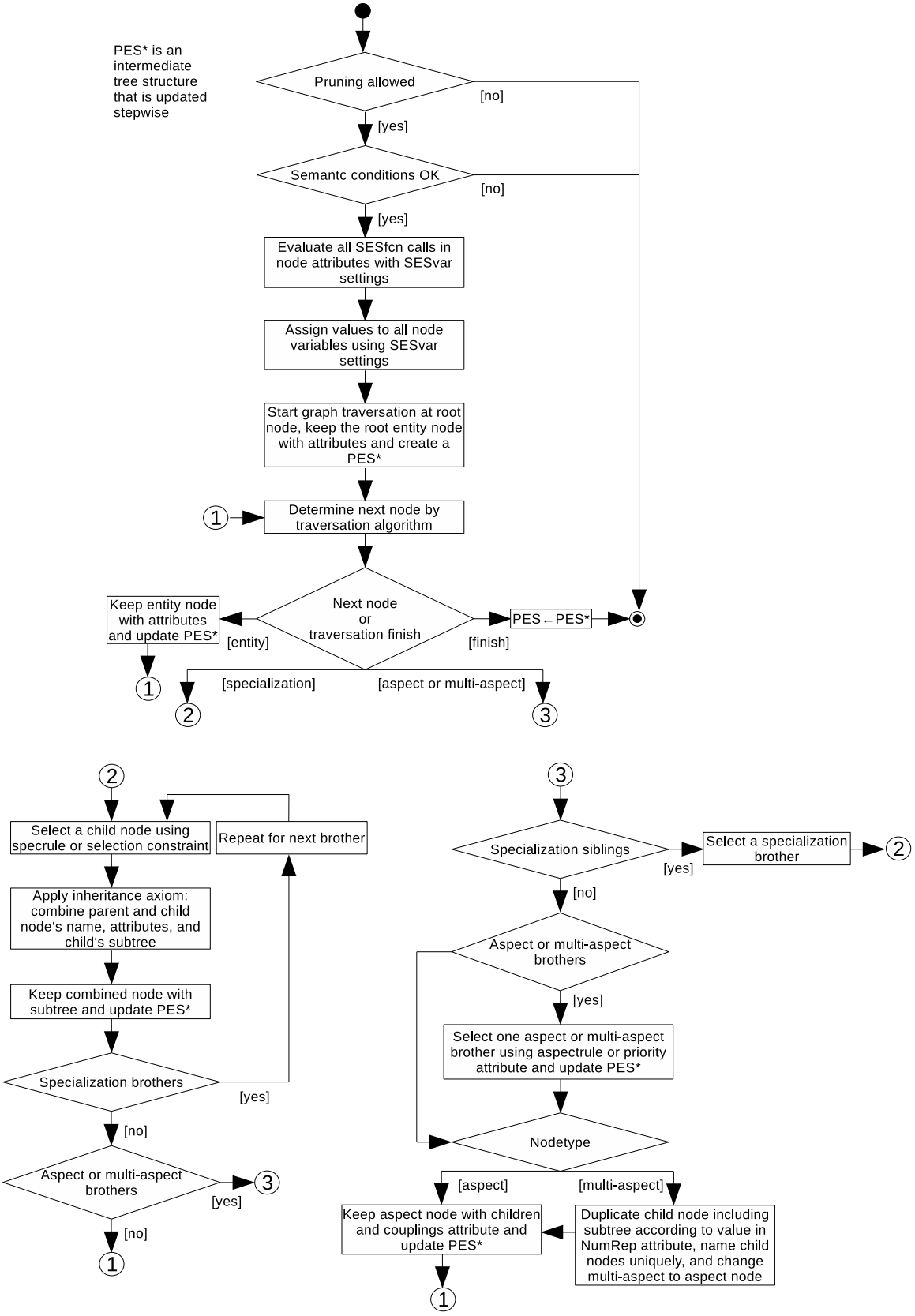


Figure 3.2: Essential node type related pruning steps without details of graph traversal.

Aspect nodes
When pruning an aspect node, sibling relationships must first be checked. Brothers of type specialization need to be pruned first as discussed before. If brother nodes of type aspect or multi-aspect exist, a total of one aspect or multi-aspect node must be selected. On the one hand, aspect or multi-aspect nodes can already be brothers in the SES, on the other hand these nodes can have become brothers by resolving a specialization node. In the first case, the node is selected by the aspectrules as given in the SES. In the second case, the node is selected using the priority attribute. Unselected nodes of type aspect or multi-aspect in a sibling relationship are deleted including their subtrees. A selected aspect node is not modified. Its couplings attribute and parent-child node relationships remain unchanged.

Multi-aspect nodes
Multi-aspect nodes are a special type of aspect nodes. The pruning of the SES is done up to the reduction to exclusively one sibling node as with aspect nodes. Additionally, a multi-aspect node is converted to an ordinary aspect node during pruning. For this purpose the NumRep attribute of the multi-aspect node is evaluated. According to its value, the child node including its attributes is duplicated. When duplicating, a unique number is added to the names of all child nodes to comply with the axiom of valid brothers. Finally, the name and the couplings attribute of the converted node is adjusted with respect to the number and names of the generated children.

Flattening The result of the pruning operation is a decision-free tree, called PES, which describes a unique system configuration of a selected candidate model. A PES can consist of several hierarchy levels. In a PES as specification of a modular-hierarchical system configuration in the sense of M&S, the root node describes the overall model, inner entity nodes structures of coupled systems, and leaf nodes links to dynamic basic models in an MB. For goal-oriented automated simulation experiments usually only the simulation results are of interest and not the modular-hierarchical model structures of all investigated variants of a FoM. Under this objective it makes sense to resolve the hierarchy levels of the PES before generating a simulation model. The resolution of the hierarchy levels is called *flattening*. Flattening simplifies the implementation of the build operation to generate simulation models and reduces the execution time of simulation models, even though simulation environments usually check models for structural simplifications before simulation. The resulting tree structure after flattening is called *Flattened Pruned Entity Structure* (FPES). A minimal FPES consists of three layers, beginning with the root node, followed by an aspect node with a couplings attribute and all leaf nodes of the former PES in one layer. In the context of the architecture to be realized, a flattening operation was implemented. Applying the flattening method to a PES, all inner nodes are removed. Resolving the inner nodes means that the coupling relationships, which are now all specified in the couplings attribute of the one remaining aspect node, must be adjusted.

In addition to the transformation of the linkage relationships, the node names must be checked in the FPES. In the PES, the same node names can occur in different paths of the tree. During flattening, the paths are resolved and all leaf nodes of the PES become brothers on one hierarchy level. Consequently, the axiom of valid brothers may be violated in the FPES. Therefore, the flattening operation must check all leaf nodes for name equality and rename them if necessary during the transformation. If nodes are renamed, the entries in the coupling tuples must be updated.

Build The build method creates a simulation model according to the structure and parameter information in the PES or FPES. A simulation model is composed from components of the MB, which are referenced via the *mb-attribute* in leaf entities. The individual components are parameterized and connected according to the specified coupling relationships. In particular, the types of ports are also defined in the extended SES specification and thus the type of coupling is also specified. The MB organizes simulator-specific components. The components are characterized by a simulator-specific interface, which means that their parameterization and ports are suitable for a particular simulator. This information must be stored in the attributes of the leaf nodes as well as in the couplings attributes in the SES. This contradicts the statement that an SES is simulator-independent (Zeigler et al. [167]). Thus, to achieve a fully simulator-independent representation of system configurations in the SES, components in the MB must have identical interfaces across simulators. Approaches for this are presented in Chapter 4.

Generalized the build method can be applied to component-based systems of any type. Thus it is possible, for example, to describe variants of a software system and to create executable code with the help of the build method. This form of software generation is briefly discussed in Section 4.4.7.

3.2.3 Demonstration of SES/MB-based Modeling

To further explain the advanced concepts of SES/MB-based modeling, the modeling of the feedback control system example introduced in Section 2.5.1 is considered. The concepts shown here are based on Schmidt [135] and Pawletta et al. [114],[113]. They form the foundation for further work by the author. Variant modeling with an SES, the construction of a simulator-specific MB, the selection of a candidate model by automated pruning, flattening, and the generation of executable simulation models are discussed. Figure 3.3 shows the modeling of the model variants of the feedback control system with an SES. The SES models two structural variants of the feedback control system: (i) without feedforward and (ii) with feedforward. By assigning values to the SESvar *feedforward*, which defines the input interface of the SES, a variant can be selected automatically. The semantic condition defines the allowed value assignment of the SESvar. In addition, the system can be parameterized differently according to Section 2.5.1. The modeling of different parameterization variants is trivial for the example and was therefore not specified in the SES for reasons of overview.

The root node *ctrlSys* represents the overall system and the subsequent aspect node *ctrlSysDEC* with its composition of subsystems. The overall system *ctrlSys* consists of the process *procUnitSys*, a controller *ctrlPIDSys*, the setpoint system *sourceSys*, and the disturbance system with *sourceDist* element and transfer function *tfDist*. For the correct composition of the subsystems the components adder *addDist* and subtract *feedbackSys* are also required. The coupling relations for the composition of *ctrlSys* from the subsystems are specified in the attribute couplings of the aspect node *ctrlSysDEC*.

The variation point for mapping the two structure variants, *ctrlSys* without or with *feedforwardCtrl* subsystem, is modeled by the specialization node *feedforwardCtrlSPEC*. According to Deatcu et al. [40], options can be expressed by a specialization associated with the NONE node. Therefore, the *feedforwardCtrlSPEC* node has children *fc* and

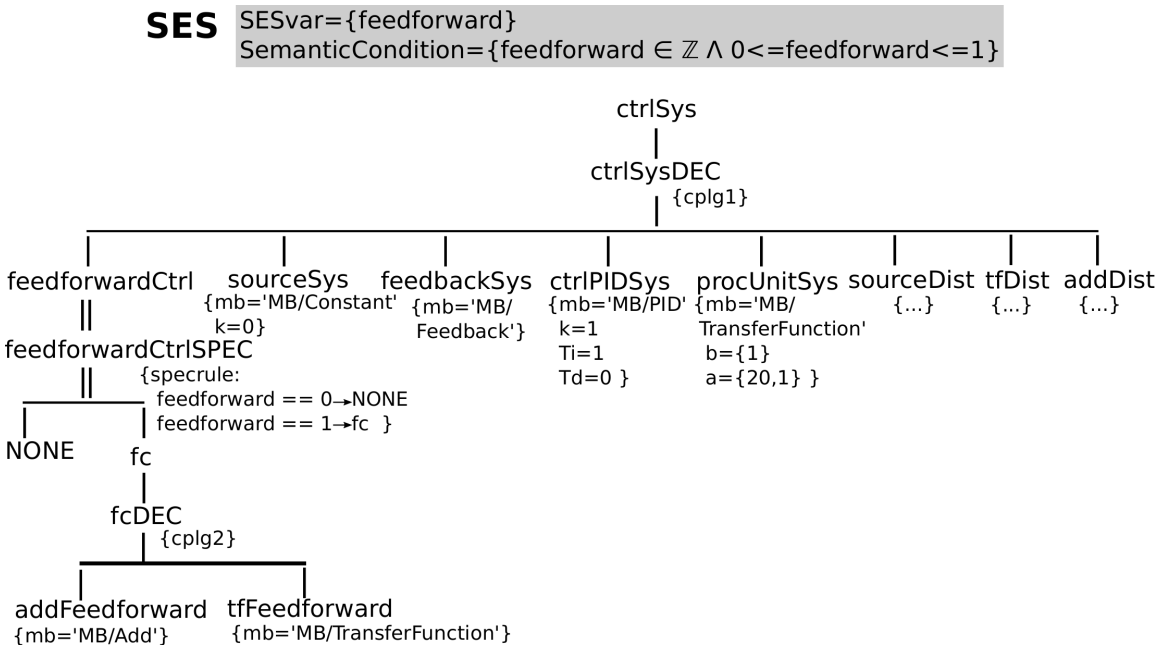


Figure 3.3: SES of the feedback control system.

NONE. The structure of the feedforward controller is described by the subtree of the entity node *fc*. An *fc* consists of a transfer function *tfFeedforward* and an adder system *addFeedforward*. The coupling relations are defined in the attribute *cplg2* of the aspect node *fcDEC*. The couplings are presented in Table 3.1.

Table 3.1: Couplings in *cplg2* in the SES of the feedback control system.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
fc	u1	SPR	tfFeedforward	u	SPR
tfFeedforward	y	SPR	addFeedforward	u1	SPR
fc	u2	SPR	addFeedforward	u2	SPR
addFeedforward	y	SPR	fc	y	SPR

The conditions for selecting a structure variant during pruning are defined in the attribute *specrule* of node *feedforwardCtrlSPEC*. The *specrule* is based on the SESvar *feedforward* and controls whether a candidate model with or without feedforward system is selected. Due to the different model structures of *ctrlSys* – with or without subsystem *feedforwardCtrl* – the coupling relations in node *ctrlSysDEC* are structure dependent. Therefore, the couplings in the attribute *cplg1* of *ctrlSysDEC* are defined using an SESfcn that describes variable couplings depending on the SESvar *feedforward* in addition to fixed couplings. This SESfcn *cplfcn* is shown in Listing 3.1 in pseudocode. Next to the SESvar *feedforward* the special variable *children* is specified, which refers to the names of the child nodes of *ctrlSysDEC*. Using the variable *children* in an SESfcn decouples node names in the SES from the representation of node names in the SESfcn. In contrast to the *cplg1* defined with an SESfcn, the couplings in *cplg2* are defined as fixed as presented in Table 3.1.

```
cplfcn(feedforward, children):
    #children(0) is feedforwardCtrl
    #children(1) is sourceSys
    #children(2) is feedbackSys
    #children(3) is ctrlPIDSys
    #children(4) is procUnitSys
    #children(5) is sourceDist
    #children(6) is tfDist
    #children(7) is addDist

    #fixed couplings
    cplg(1)=(children(1), y / SPR, children(2), u1 / SPR)
    cplg(2)=(children(2), y / SPR, children(3), u / SPR)
    cplg(3)=(children(4), y / SPR, children(7), u2 / SPR)
    cplg(4)=(children(7), y / SPR, children(2), u2 / SPR)
    cplg(5)=(children(5), y / SPR, children(6), u / SPR)
    cplg(6)=(children(6), y / SPR, children(7), u1 / SPR)

    #variable couplings
    if feedforward==0:
        cplg(7)=(children(3), y / SPR, children(4), u / SPR)
    else if feedforward==1:
        cplg(7)=(children(5), y / SPR, children(0), u1 / SPR)
        cplg(8)=(children(3), y / SPR, children(0), u2 / SPR)
        cplg(9)=(children(0), y / SPR, children(4), u / SPR)
    end

#return
return(cplg)
```

Listing 3.1: Couplings in *cplg1* in the SES of the feedback control system.

All leaf nodes have an *mb-attribute* that refers to a basic model in an MB. For example, the *ctrlPIDSys* node refers to the basic model *PID* and defines the configuration parameters $k = 1$, $Ti = 1$ and $Td = 0$ for it. Alternative parameterizations can be specified in the attributes of the leaf entities. The selection of a parameterization variant during automated pruning can be described analogously to variable coupling relationships with an SESfcn or using multisets (Pawletta et al. [111]).

The selection of a system variant is done with the help of pruning. Pruning evaluates the value of the SESvar *feedforward*. The only decision node in this example is the *feedforwardCtrlSPEC* node. According to the pruning algorithm in Section 3.2.2, the specialization is resolved and the selected child node is unified with the parent node of the specialization. Both structure variants are shown in Figure 3.4. The SESfcn *cplfcn* in Listing 3.1 for calculating the couplings for the *ctrlSysDEC* node is evaluated so that all variable coupling relationships of the SES are defined fixed. For the variant with feedforward control, it is necessary to adjust the couplings in the PES according to changed node names during pruning. Some couplings of PES number 2 in Figure 3.4 are presented in Table 3.2. A complete list of the couplings is presented in Appendix C.

As explained in Section 3.2.2, converting the PES to an FPES facilitates model generation. Figure 3.5 shows the FPES for PES number 2 in Figure 3.4.

Flattening requires the adaptation of the couplings, because inner nodes are removed. This is shown by an example. In Table 3.2 two couplings of PES number 2 refer to the inner node *fc_feedforwardCtrl* in the PES. The combination of entity name, port, and type *fc_feedforwardCtrl / y / SPR* is found in two couplings as source and as sink. These two couplings can be united by replacing *fc_feedforwardCtrl*. The resulting coupling refers to leaf nodes. In addition to the first two couplings taken from the PES the resulting coupling is shown as third coupling in Table 3.3. A complete list of the couplings is presented in

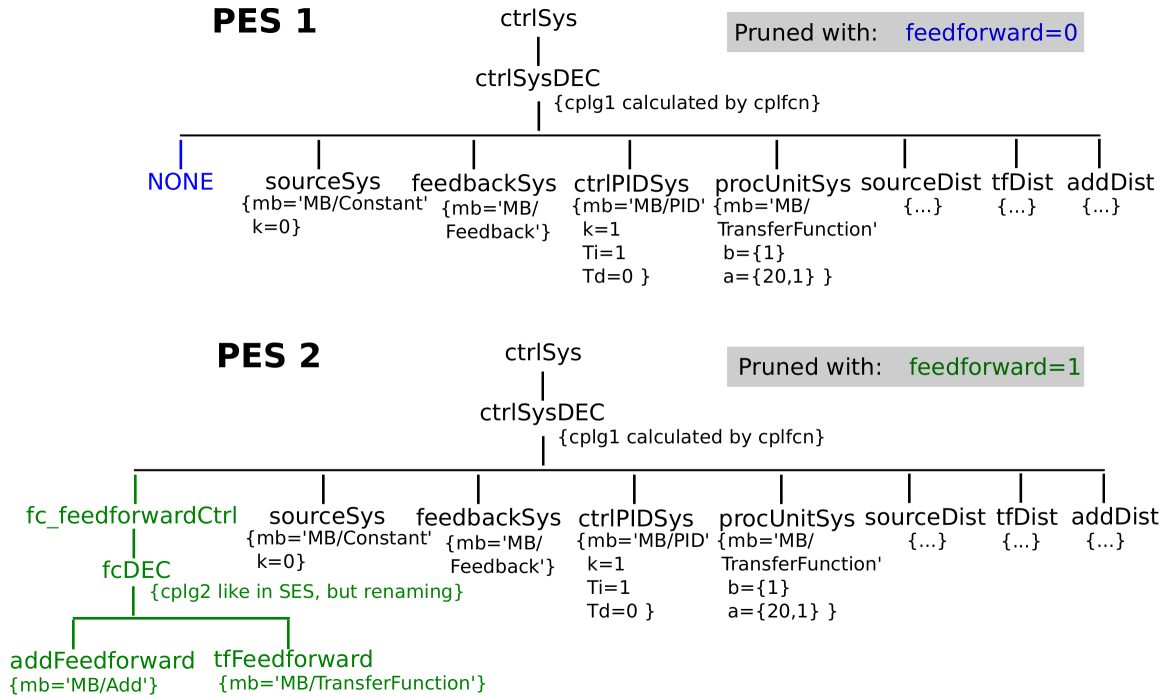


Figure 3.4: Possible PES pruned from the SES.

Table 3.2: Some couplings of the PES number 2 in Figure 3.4.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
			cplg1		
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
fc_feedforwardCtrl	y	SPR	procUnitSys	u	SPR
...	...	...	...	...	...
			cplg2		
addFeedforward	y	SPR	fc_feedforwardCtrl	y	SPR
...	...	...	...	...	...

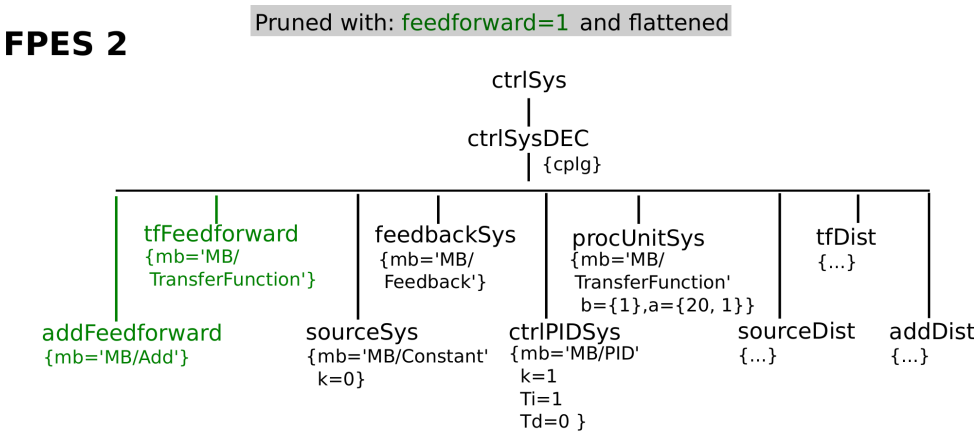


Figure 3.5: FPES for PES number 2 in Figure 3.4.

Appendix C.

Table 3.3: Some adapted couplings in the FPES of the feedback control system.

Source node			*Sink node*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
addFeedforward	y	SPR	procUnitSys	u	SPR
...	...	...	...	...	...

The MB with basic models is created directly from basic components of the general Modelica library [91]. Figure 3.6 shows the basic models of the MB of the feedback control system.

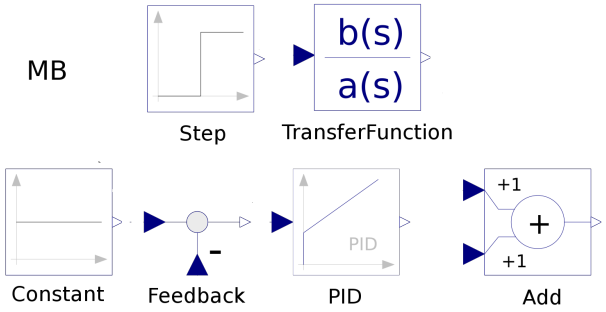


Figure 3.6: MB with Modelica basic models.

The build operation is implemented simulator-dependent for generating Modelica models. The generated models can be executed with Modelica simulators such as OpenModelica or Dymola. Figure 3.7 shows the two possible model structures, each of which can be generated based on an FPES.

3.2.4 Automated Pruning of Hierarchical Multi-Aspects

So far, the descriptive node types aspect and specialization have been considered primarily. Compositions were specified by aspect nodes and decision points with specialization nodes. The multi-aspect node was introduced only in principle. Now the multi-aspect node will be considered in detail under the requirements of the automated pruning of an SES. For explanation, an example is developed step by step.

Figure 3.8 shows a minimal SES consisting of two entity nodes and one multi-aspect. The root node *lab* specifies a computer lab consisting of a set of similar networked computers (*c_1, c_2, ...*). The multi-aspect node *csMASP* specifies the composition of the entity *lab* from similar entities of type *c*. A multi-aspect node defines two attributes: (i) NumRep and (ii) couplings. The NumRep attribute defines how many entities of the type of the following entity, in this case how many computers of type *c*, are to be generated when pruning the SES. Due to the generation property, the subsequent entity node is called generating entity. In the shown SES the NumRep attribute is defined with the SESvar *NumC*. The couplings attribute specifies the coupling relations of the entities to be generated with

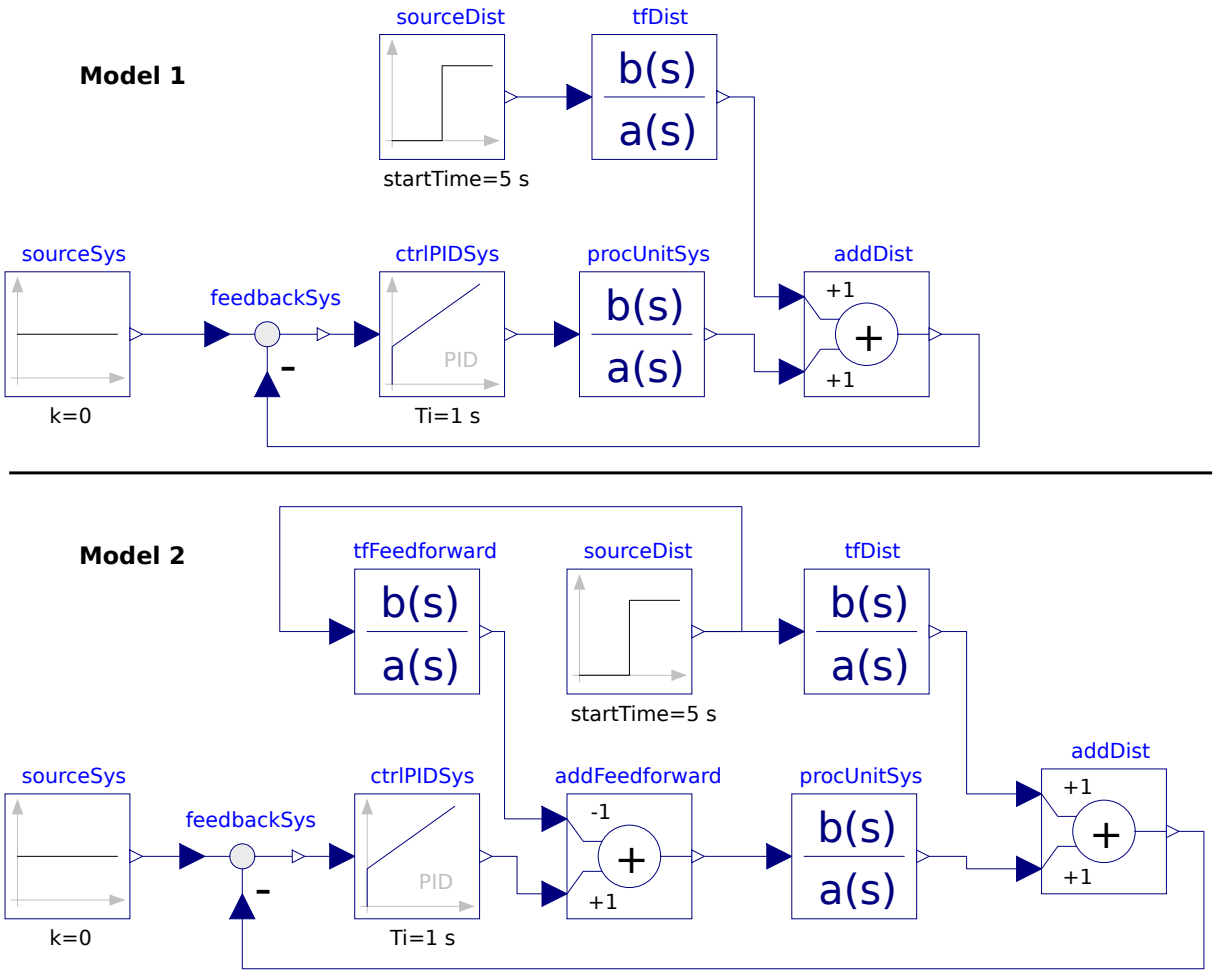


Figure 3.7: Modelica-based model structures generated with the build operation and using an FPES: (i) case *feedforward* = *0* → without feedforward control and (ii) case *feedforward* = *1* → with feedforward control.

each other and with the parent entity. The couplings attribute for the specification of the networking of the computers is not specified here. Since the same laws apply to the couplings attribute as discussed for the aspect node, they are not explicitly considered below. The coupling relationships, which are often variable depending on the NumRep attribute, can be described compactly with SESfcns.

The right part Figure 3.8 shows a PES derived by pruning, assuming a value assignment $NumC = 3$. When pruning a multi-aspect node, it is transformed into an aspect node. The generated entities receive an index, so that the axiom *valid brothers* is obeyed.

Figure 3.9 shows an extension of the SES with an entity node *campus* and a multi-aspect named *labsMASP*. This SES describes a campus that hosts multiple computer labs, each with multiple computers. The occurrence of multiple multi-aspects in a path is called hierarchical multi-aspects. The derivation of a PES shown in two steps shows that this SES only allows the derivation of labs with the same number of computers.

The approaches to automated pruning of SES known so far all introduce restrictions on the use of multi-aspects. Schmidt [135] limits the use in that the child node of a multi-aspect must be a leaf node. Accordingly, he prohibits hierarchical multi-aspects

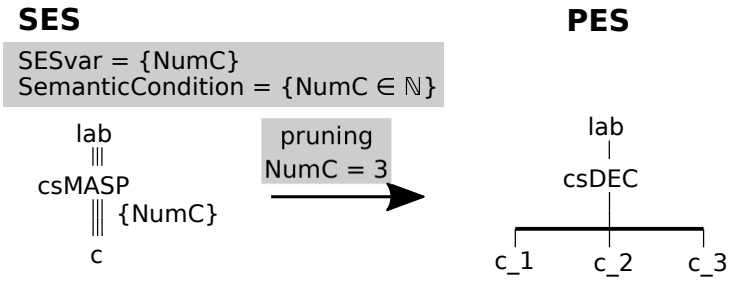


Figure 3.8: SES with one multi-aspect and derivation of a possible PES.

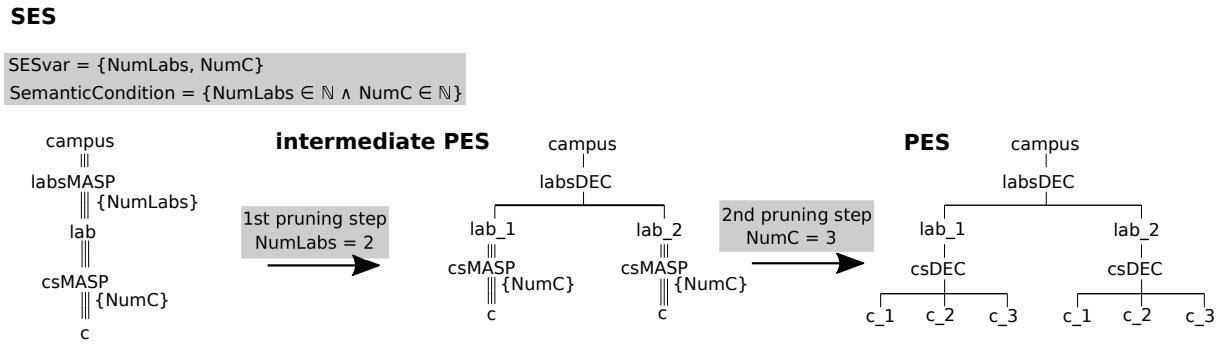


Figure 3.9: SES with two multi-aspects in one path and derivation of a possible PES by pruning.

completely. Zeigler and Sarjoughian [165] support hierarchical multi-aspects in the MS4Me software system analogous to the example in Figure 3.9. It shall be criticized that there is no support for specifying multiplicity in the NumRep attribute.

Figure 3.10 shows a new approach for specifying multiplicity in the NumRep attribute, which was first published by Folkerts et al. [53]. The entity node following a multi-aspect is called generating entity. In the new approach, the generating entity defines a variable of the same name with underscore prefix, in this case named *_lab*. This variable is automatically assigned the suffix index of the generated entity as a value during pruning. Subsequent nodes in the same path can evaluate this special attribute. In the example in Figure 3.10, the *_lab* attribute is used by the multi-aspect *csMASP*. When pruning *csMASP*, *_lab* is passed as input argument to the SESfcn *fun*, which returns the current value of the NumRep attribute of *csMASP* depending on *_lab*. The example shows that for the specification of multiplicity only the SESfcn *fun* and the vectorial SESvar *Computers* must be defined.

The introduced extension for the specification of multiplicity can be used in combination with specializations. Figure 3.11 shows an example. As before, the SES describes a campus with computer labs whose number is defined by the SESvar *NumLabs* in the NumRep attribute of the multi-aspect *labsMASP*. The generating entity *lab* is followed by a specialization *labSpec*. This specialization decides in the selection rule attribute whether a lab with desktop computers *ds* (left path) or a lab with notebooks *ns* (right path) is to be generated. In addition to the specialization, an SESvar *Types* and an SESfcn *fun1* are introduced. The SESfcn *fun1* works analogous to the SESfcn *fun*. During pruning, the computer type to be generated is determined by calling *fun1(_lab)* in the selection rule of *labSpec* on the basis of the index of a generated *lab* entity. Figure 3.11 shows the derivation of a PES for the following SESvar assignment:

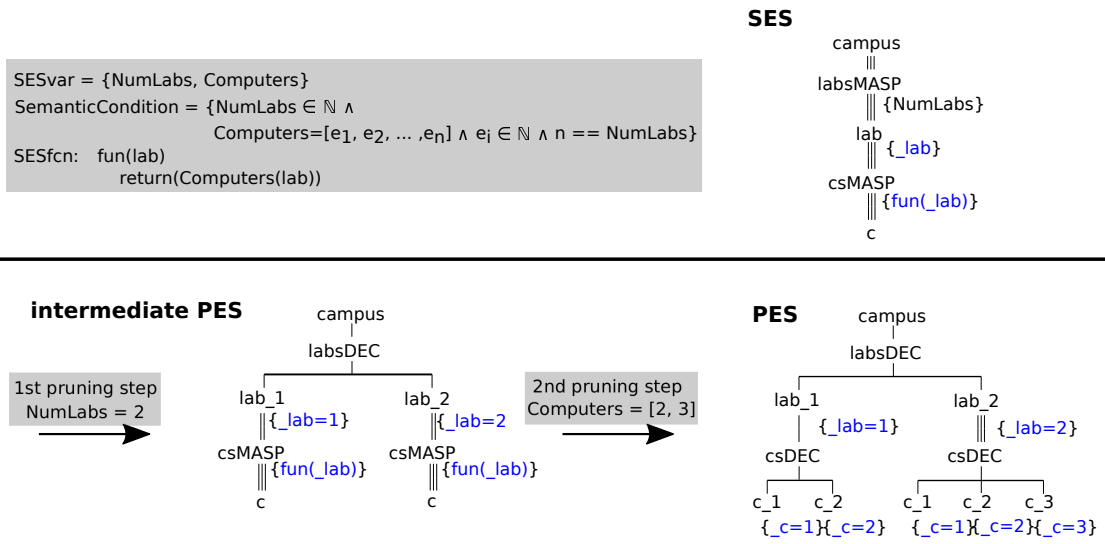


Figure 3.10: SES with two multi-aspects in one path and specification of multiplicity as well as derivation of a possible PES by pruning.

$NumLabs = 2$, $Types = [\,'d','n'\,]$, and $Computers = [\,3,4\,]$.

The variables encode the generation of two labs, one with three desktop computers and one with four notebooks.

More details about the developed extension and its usage to specify a family of systems in the context of an engineering application are published by Folkerts et al. in [53].

3.2.5 Summary of New Extensions

The extensions of the SES developed in the course of this work are briefly summarized here. These are the extension of couplings with port types, the extended pruning operation, and the extension regarding hierarchical multi-aspect nodes.

Basic models of different type differ in their interface depending on their dynamic behavior and their purpose. While in the MB basic models of different types can be organized, in the SES the type of a basic model must be taken into account. Different types of basic models can be referenced with the *mb-attribute*. However, with the aim to support several simulation tools in the SES the interface of basic models needs to be specified in couplings in the SES. This allows the description and generation of models of different type or even hybrid models, which are composed using basic models of different system classes, with the SES/MB approach.

A central aspect is the selection of one model variant with the pruning method. Describing complex systems often requires greater hierarchy depths according to multi-aspect nodes and the combination of different descriptive nodes on one level. The *Cyber-Physical Systems* (CPS) motivated in the introduction often represent complex systems. The extension of the pruning algorithm is thus an essential development.

Another extension was discussed in Section 3.2.4. Multi-aspect nodes are a powerful modeling element. Using multi-aspect nodes the description of complex systems is simplified. However, pruning SES with multi-aspect nodes without user interaction is challenging

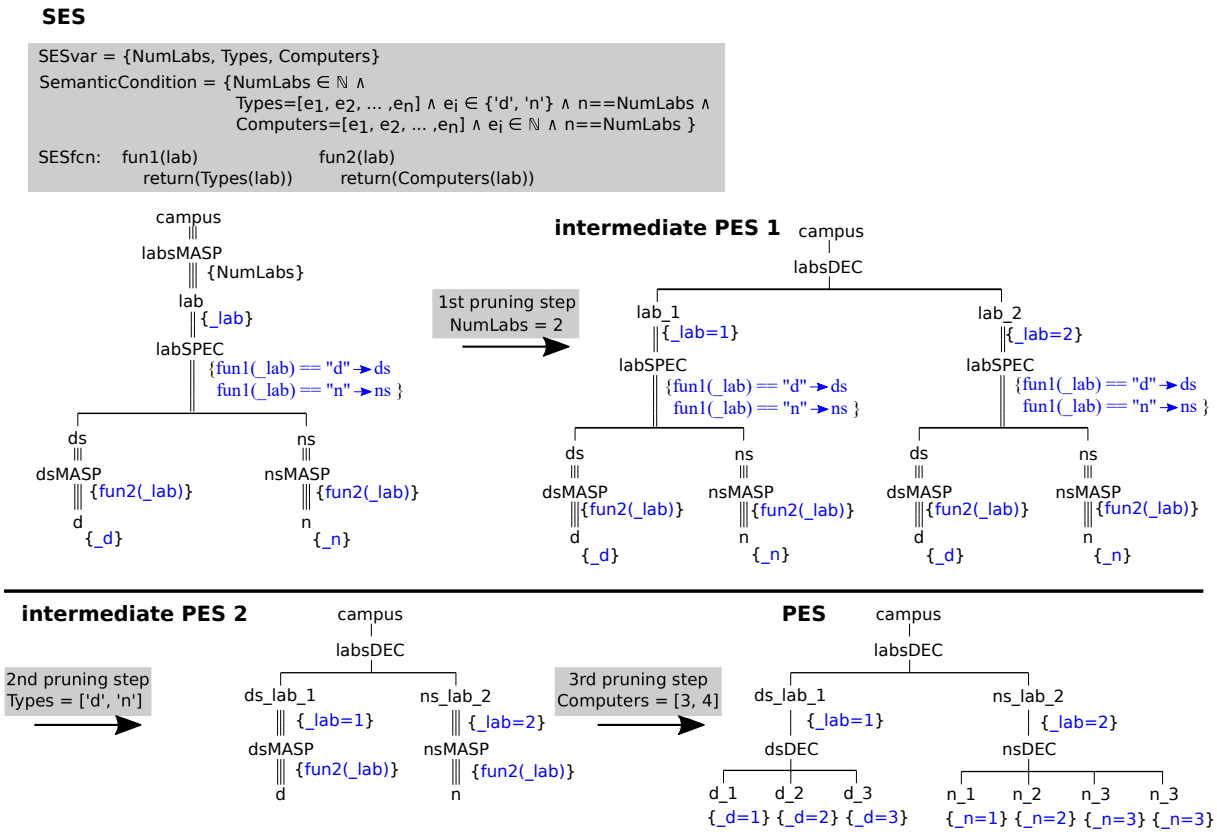


Figure 3.11: SES with hierarchical multi-aspects in combination with a specialization and derivation of a possible PES by pruning.

(Zeigler and Hammonds [164] and Zeigler and Sarjoughian [165]). This is even more true if the multi-aspect nodes are combined with specialization nodes or if there are multiple multi-aspect nodes in one path. Unlike the other descriptive nodes, multi-aspect nodes generate new nodes during pruning. With the introduction of a new attribute at generating nodes, the pruning can be automized for complex SES.

3.3 Linking With Simulation Experiments

In this subsection, concepts for linking the SES/MB-based modeling approach with simulation experiments are discussed. In the introduction to Chapter 2, it is noted that each *simulation experiment* consists of a *model* and a *simulator* to calculate the dynamic behavior over time. The one-time calculation of the dynamic behavior is generally referred to as a *simulation run* or simply *simulation*. In the early years of M&S, simulation models and experiments were coded by hand in assembler or FORTRAN and no explicit distinction was made between model, simulator, and experiment. With the advent of specialized program systems in the 1960s (Gordon [60]) the separation into model and simulator became established. With the concept of the *Experimental Frame* (EF) Zeigler introduced in [162] an interface to separate the dynamic model from the application context. The EF specifies in this sense an interface for a model to perform experiments with it. The concept of the EF and its connection with the SES/MB approach are considered in the first two subsections.

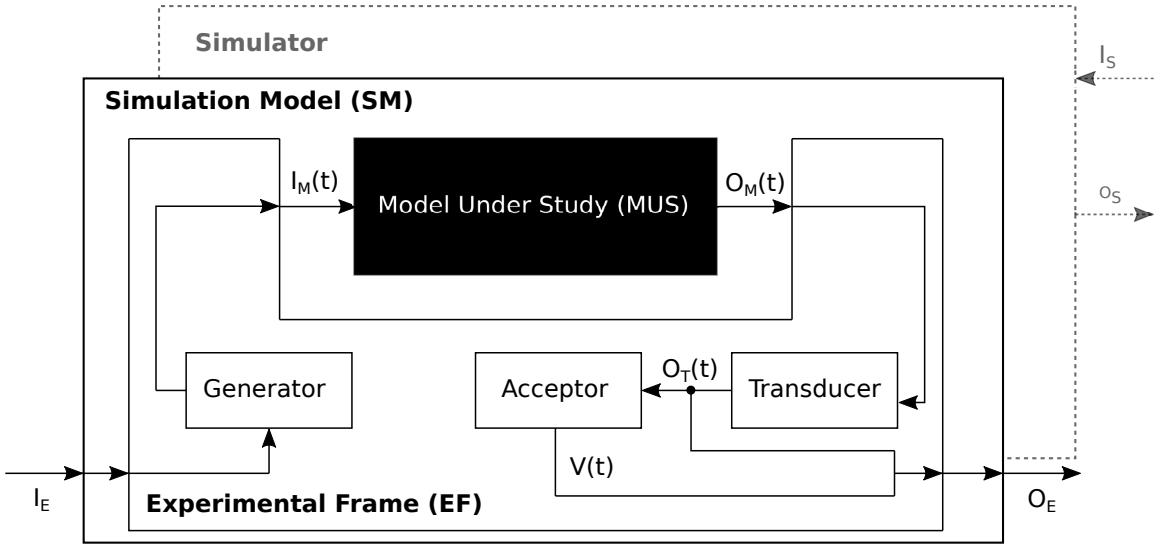
The life cycle model of a FoM for simulation in Figure 2.1 in Section 2.1 states, that a simulation experiment usually does not consist of a single simulation run. M&S is often used in combination with other numerical methods (Barton [7], Kleijnen [76], Kleijnen [75], Leye [83]), such as numerical optimization methods (Carson and Maria [25]). As an umbrella term for numerical experiments with integrated simulation runs, the term *simulation-based experiments* is often used in the M&S community. Based on a broad literature review, Schmidt [135] classifies simulation-based experiments with respect to the *experiment objectives*, the *experiment phases*, and the *experiment setup*. For the experiment objectives he distinguishes five categories, such as parameter screening, sensitivity analysis, or optimization. By experiment phases he understands a sequence of experiment objectives in a simulation study. He examines the experimental setup in analogy to model structures and establishes a connection with the SES/MB approach. In the context of this work the investigations to the experiment structure and the connection with the SES/MB approach are of interest, which is dealt with in the third subsection. A classification of the focus of this work, the experimentation with different simulators, is given before specific challenges in this regard are discussed.

3.3.1 Experimental Frame as Interface to Simulation Experiments

A simulation experiment uses a simulation model in a specific context. It defines certain experimental conditions and experimental objectives. Zeigler introduced in [162] the concept of the EF to represent the context of a model. In general, an EF is a layer surrounding the source system, which is in our domain the model. The concept of EF is continuously further developed, researched, and adapted by several authors. Zeigler [163] and Rozenblit [127] adapted the EF for hierarchical *Discrete Event System Specification* (DEVS) modeling and used it in combination with the SES methodology. Further refinements and applications of the concept of EF are published by Zeigler et al. [167], Daum and Sargent [39], Traoré and Muzy [146], Denil et al. [41], and Schmidt [135].

In summary, it can be stated that an EF can be implemented using the three components: *generator*, *transducer*, and *acceptor*. However, all three components do not always have to be present. An EF is a coupled system, whose components can be structured hierarchically. Since the EF is part of the *Simulation Model* (SM), the model for which the EF defines the context is often referred to as *Model Under Study* (MUS) (Schmidt [135]). The coupling relations of the components of the EF and the EF with the MUS are not defined. An SM is a coupled system on the highest hierarchical level, consisting of an EF and a MUS. The EF generates inputs to a MUS and processes outputs of a MUS. Since the EF interfaces to the simulation-based experiment, it defines the input/output interface of an SM according to Traoré and Muzy in [146]. An EF can be valid for different MUS. On the other hand, different EFs can be specified for one MUS. Figure 3.12 shows the block diagram of an SM with EF and MUS. Since a simulator is necessary as an execution component for each simulation-based experiment, it is shown in the background.

Figure 3.12 shows exemplary coupling relations of the EF. The input quantities I and output quantities O, which refer to the EF are marked by the index E, those which refer to the MUS with the index M. The quantities I_E and O_E are not time dependent. At the start of the simulation the EF receives input quantities of the experiment to be executed



I_E...input variables of EF; $I_M(t)$...time-dependent input variables of MUS; $O_M(t)$...time-dependent output variables of MUS; I_S...input variables of simulator; O_S...output variables of simulator; $O_T(t)$...time-dependent outcome measures; V(t)...time dependent verdict; O_E...output variables of EF

Figure 3.12: Block diagram of a simulation model with EF and MUS as well as executing simulator according to Schmidt [135].

and after completion of the simulation results are returned to the experiment in the form of output quantities by the EF. In contrast, the input quantities $I_M(t)$ and output quantities $O_M(t)$ of the MUS are time-dependent.

The generator of the EF configures the MUS. The configuration may involve setting initial values at the start of the simulation or computing time-dependent input quantities of the MUS during the simulation. The time-dependent output quantities of the MUS are processed by the transducer. The latter computes the experiment-dependent values of interest from the $O_M(t)$. The acceptor receives the outputs from the transducer and checks for compliance with experiment conditions. Of course, the acceptor could directly evaluate output quantities of the MUS via an appropriate coupling relation. In case of non-compliance with given experiment conditions or if the study goal is reached, the acceptor aborts the simulation.

Analogous to the specification of system variants of a MUS, variants and compositions of EFs, and compositions of SMs from MUS and EFs can be described with an SES. Basic models for the composition of an EF can be organized in an MB. This will be demonstrated by an example in the next subsection.

3.3.2 Linking SES/MB-based Modeling and the Concept of the Experimental Frame by Means of an Example

To demonstrate SES/MB-based modeling including the use of discussed extensions, a feedback control system was modeled with two different model structures in Section 3.2.3. Based on the SES and MB specified there, the example is extended here to include an EF. For this purpose, the context of the model is defined first.

Figure 3.7 shows the two model structures specified in the SES in Figure 3.3. The EF is intended to support the investigation of both model structures. The parameters k and T of the PID controller, i.e. the basic system *ctrlPIDSys*, shall be variably configurable. Furthermore, different disturbance signal sources, named *sourceDist* in Figure 3.7, shall be investigable. The values of interest are the maximum overshoot of the controlled variable, the settling time as well as the course of the controlled variable over time. The EF shall define boundary conditions on the basis of maximum permissible values for overshoot and settling time. If a boundary condition is violated, the simulation run shall be aborted and judged as invalid. If there is no violation of the boundary conditions, the simulation run shall be terminated by the simulator in a regular way.

Figure 3.13 shows the structure of an SM consisting of an EF and the model structure of the feedback control system without feedforward control, named *ctrlSys*, as MUS. In contrast to the model structure in Figure 3.7, the disturbance source *sourceDist* is no longer part of the MUS, but part of the EF. Moreover, the parameters k and T of the PID controller *ctrlPIDSys* are set via external input couplings. The EF is composed by a *generator*, *acceptor*, *transducer*, and a *mux* component. The generator is a coupled model and the other ones are basic models. The *generator* gets the current input variables $I_E(t_0)$:

- k, T the PID controller parameters,
- *dist* the parameters for the disturbing source,
- y_{acc}^{max} the maximal acceptable control deviation, and
- t_{cs}^{max} the maximal acceptable settling time after a disturbance

for a simulation run at simulation start time. At time t_0 the generator initializes components of the EF and the MUS via coupling relations. During a simulation run, the generator's disturbance source subsystem generates inputs for the MUS *ctrlSys*. Of course, it is possible to perform more complex operations in a generator as demonstrated by this example. The time dependent values of the control variable *y(t)* and the reference value from the *sourceSys* are sent from the MUS to the *transducer*. The transducer stores the temporal course of the controlled variable and calculates the *maximum overshoot* and the *settling time*. The *acceptor* checks the two boundary conditions to be met on the basis of the values received from the transducer and terminates the simulation run if a boundary condition is violated and judges it with the verdict *V='false'*. If the boundary conditions are met, the verdict *V='true'* is set at the end of the simulation. At the termination or end time of the simulation, both are named with t_{final}, the acceptor sends the verdict V to the transducer, whereupon the transducer outputs the stored course of the controlled variable *y(t)* as vectorial quantities (t_{out}, Y_{out}). The basic system *mux* generates the total output $O_E(t_{final})$ of the EF.

Figure 3.14 shows a basic SES that specifies the extension of the two model structures of the feedback control system *ctrlSys* by an EF, named *ef*. The EF specifies two structure variants with the specialization node *sourceDistSPEC*. Depending on the value assignment of the SESvar *dist*, either a step function *step* or a pseudo random binary sequence *prbs* is selected as disturbance source during pruning by the *specrule*. The model structure shown in Figure 3.13 corresponds to the result of a pruning with the SESvar assignment:

$feedforward = 0$, $dist =' step'$.

For the generation of an executable SM according to Figure 3.13, the MB must be extended by the basic models for the EF referenced at the leaf nodes of the SES. Schmidt

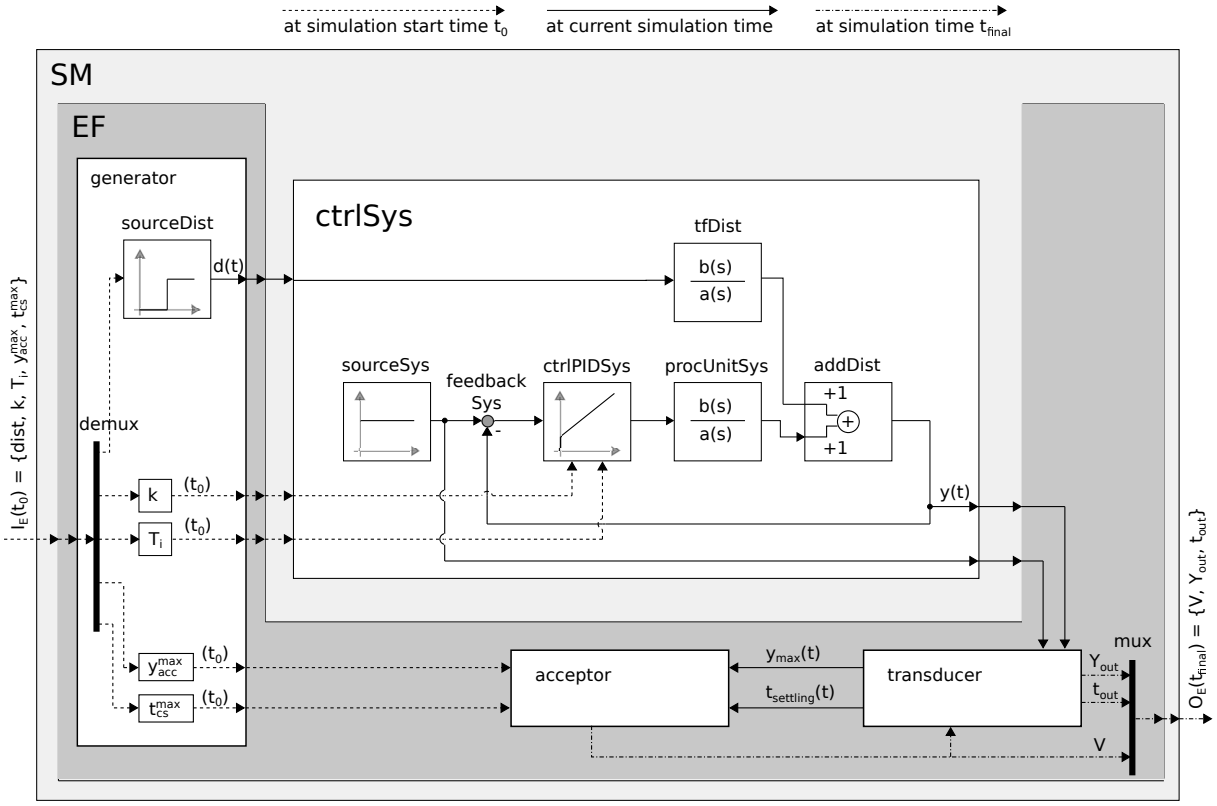


Figure 3.13: Structure of an SM consisting of an EF and the model structure of the feedback control system without feedforward control as MUS.

et al. show in [137] an application in the field of model-based testing.

An EF specifies only the interface for simulation-based experiments with a MUS. The execution of an experiment requires a specification of the experiment procedure and a link to execution methods. This will be discussed in the next subsection.

3.3.3 Structure of Simulation-based Experiments

The minimum simulation-based experiment is a single simulation run. To execute a single simulation run, simulation parameters such as *start time*, *final time* or, in the case of a continuous simulation, a *step size* or methods such as a *step size method* and an *Ordinary Differential Equation (ODE) solver method* must be defined, and a connection with a *simulator* must be established. A first approach to the systematic structuring and execution of simulation experiments was proposed by Breitenecker [20] for continuous system simulation with the *Continuous System Simulation Language* (CSSL) standard (Strauss et al. [140]). According to Breitenecker [20], a decisive next step was the structuring of simulation-based experiments by Zeigler [162] and Zeigler [163] in: *model frame*, *experimental frame*, and *experiment control* as well as the connection of the M&S with an MB (Zeigler [163]), whereby Breitenecker [20] does not deal with the SES. Breitenecker [20] criticises the babylonian confusion of language in M&S due to not clearly defined terms and defines the broadly used terms: *model*, *method*, *experiment* as follows:

- A *model* is the description of a process using a mathematical formulation and a

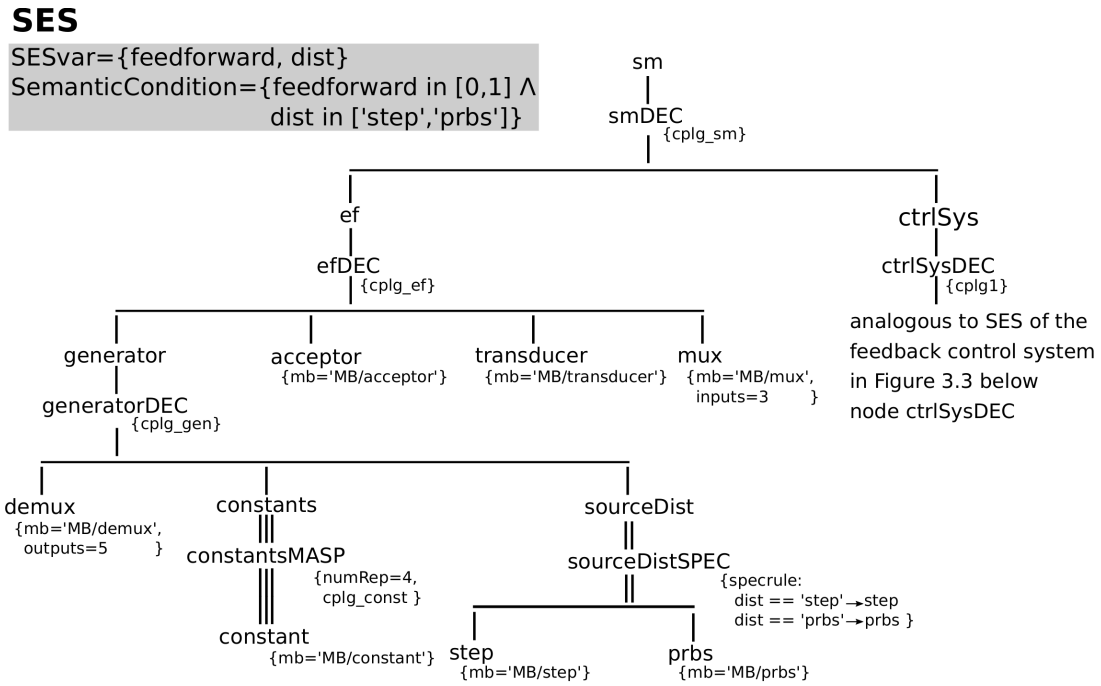


Figure 3.14: SES of the feedback control system as MUS and appropriate EF.

certain language.

- A *method* is a procedure, an algorithm, which does anything with the model.
- An *experiment* is the performance of a certain method with a certain model where all aspects of execution control are included.

Without explicit reference to Breitenecker, Schmidt [135] also bases his classification of the structure of simulation-based experiments on the triple (model, method, experiment) and defines the three terms analogously to Breitenecker [20]. Starting point for Schmidt are the works of Zeigler and colleagues in [163] and [167] as well as an analysis of a large number of later works on simulation-based experiments, such as Han et al. [67], Law and Kelton [81], Leye [83], Mittal and Risco-Martín [90], Uhrmacher and Weyns [149], or Uhrmacher [148].

Schmidt [135] defines three types of simulation-based experiments according to their complexity. He distinguishes between *simple experiments*, *complex experiments*, and *highly complex experiments*. Simple simulation-based experiments include the SM with an EF and a MUS, a simulator and an *Experiment Control* (EC). The basic steps of the EC are:

- configuration of a simulation run,
- execution of a simulation run,
- data collection, analysis, and evaluation of results, and
- a decision whether to configure, execute, and analyze another simulation run or to terminate the experiment.

Complex simulation-based experiments have additionally an *Experiment Method* (ExM) and a *Simulation Method* (SnM), as shown in Figure 3.15. The input interface I_{ExM} and the EC were added by the author. The ExM is started by the EC and returns *results* to

the EC. The ExM can execute simulation runs via the SnM, analyze and save simulation results, and reactively configure and initiate new simulation runs. The ExM is defined by Schmidt [135] as a software-implemented numerical method without direct reference to an SM or simulator. ExM have method-specific parameters P_{ExM}. These are initiated by the EC via the interface I_{ExM}. According to Schmidt [135], typical examples of ExM are numerical optimization methods or methods for sensitivity analysis. The SnM handles the connection of the ExM to the SM and the executing simulator. The configuration of a simulator is usually stored in the SnM as a parameterizable method. Additionally, it defines simulation specific parameters P_{SnM}. The P_{SnM} are set via the input interface I_{SnM} or directly from the EC. For a continuous-time simulation, the SnM typically sets simulator-specific data and methods, such as an ODE solver method, a numerical step size, and a start and end time. The SnM is a kind of wrapper for a specific simulator and a communication layer between simulator as well as SM and ExM. The EC serves on the one hand to initialize an ExM as well as the SnM, on the other hand it can link different ExM according to the different experiment phases and goals in Schmidt [135].

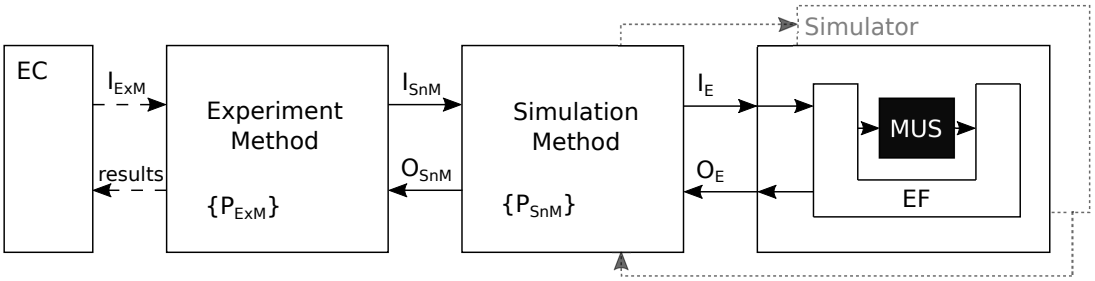


Figure 3.15: Block diagram for structuring complex simulation-based experiments according to Schmidt [135] with supplemented interface to an EC.

In complex simulation-based experiments only parameters of a MUS are varied, while in highly complex experiments different model structures and parameter variations of a MUS are studied. Schmidt [135] follows up on preliminary work by Hagendorf and colleagues (Hagendorf and Pawletta [65], Hagendorf et al. [66], Hagendorf [64]). Based on the classical SES/MB framework, Hagendorf [64] develops an approach for the combined simulation-based optimization of model parameters and model structures. Based on this, Schmidt [135] derives the class of highly complex experiments and develops his approach of highly complex experiments based on the SES/MB approach. Schmidt's [135] principal software architecture is discussed in detail in Section 4.2.

According to Schmidt's [135] approach, simulation-based experiments can be considered analogously to modular-hierarchical structured FoMs. The modularized structuring of experiments supports the implementation of basic experiment methods, which can be organized analogously to basic models in an MB. In an SES, both the structural design of a simulation-based experiment and variants of simulation-based experiments can be specified for a FoM. The formal connection with the basic experiment methods is done via the mb-attribute of experiment-related leaf nodes. By pruning the SES, a simulation-based experiment can be derived according to the structure in Figure 3.15 with the components ExM, SnM, and SM in the form of a PES or an FPES. In conjunction with an MB, a

simulator-specific experiment can then be generated using the build method.

Figure 3.16 shows a principal SES including an experiment specification for the introduced feedback control system. The root node *exp* is characterized by the aspect node *expDEC* with its attribute *cplg_exp* as a structure of SM (node *sm*), SnM (node *snm*), and ExM (node *exm*). The subtree of *sm* corresponds to the SES in Figure 3.14 and can be generated by a merge operation. The leaf entity node *snm* defines a link to a simulator-specific SnM in the MB, in this case for the simulator Simulink. Furthermore the necessary configuration parameters P_{SnM} are defined in the node attribute. The *exm* node with the following specialization node *exmSPEC* describes different experiment variants. The selection of a variant is done during pruning by evaluating the *specrule*. Depending on the selection, the following types of experiments are derived:

- selection of node *NONE* $\rightarrow$ pruning result is a *NONE* node for the ExM, which describes a simple experiment structure without an ExM. That means, all experiment steps and the inputs I_{SnM} for the SnM component are given by the EC.
- selection of node *paraStudy* $\rightarrow$ pruning result is a node *paraStudy_exm*, which describes a complex experiment structure with a basic experiment method named *paraS* as ExM. The node *paraStudy* specifies a parameter study for the MUS. The attributes could describe parameter variations for k, T, or *dist* and allow maximum values according to Section 3.3.2. The algorithm of the parameter study is coded in the experiment method *paraS*, which is stored in the MB. That means, the experiment is started by the EC, but the individual steps of the parameter study are coded in ExM *paraS*. The inputs I_{SnM} for the SnM are given by the ExM.
- selection of node *optimization* $\rightarrow$ pruning result is a node *optimization_exm*, which describes a complex experiment structure with a numerical optimization method as ExM. The SES is not further detailed here. The double edge of the node *optimization* is meant to indicate the further differentiation into different numerical optimization methods by specializations.

In the analysis of the SES in Figure 3.16, only the experiment-specific nodes *snm* and *exm* with subtree have been considered so far. According to Figure 3.3, the entity node *ctrlSys* describes two different model structures of the MUS. If the different model structures are also part of the simulation-based experiment, it is a highly complex experiment according to Schmidt's [135] classification. In the EC, the following experiment steps could be defined, with the goal of finding a controller structure that is as minimal as possible, taking into account the boundary conditions:

- define a *source disturbance sourceDist* and *experiment goals* in the form of limits to meet boundary conditions such as maximum overshoot etc.,
- try model structure *ctrlSys without feedforward control* (SESvar *feedforward = 0*) using different parameter variations (SESvar *Exppara =′ paraStudy′*),
- if the goals are reached with one of the studied configurations, then return the best configuration as overall result,
- else try model structure *ctrlSys with feedforward control* (SESvar *feedforward = 1*) using different parameter variations (SESvar *Exppara =′ paraStudy′*),
- if the goals are reached with one of the studied configurations, then return the best

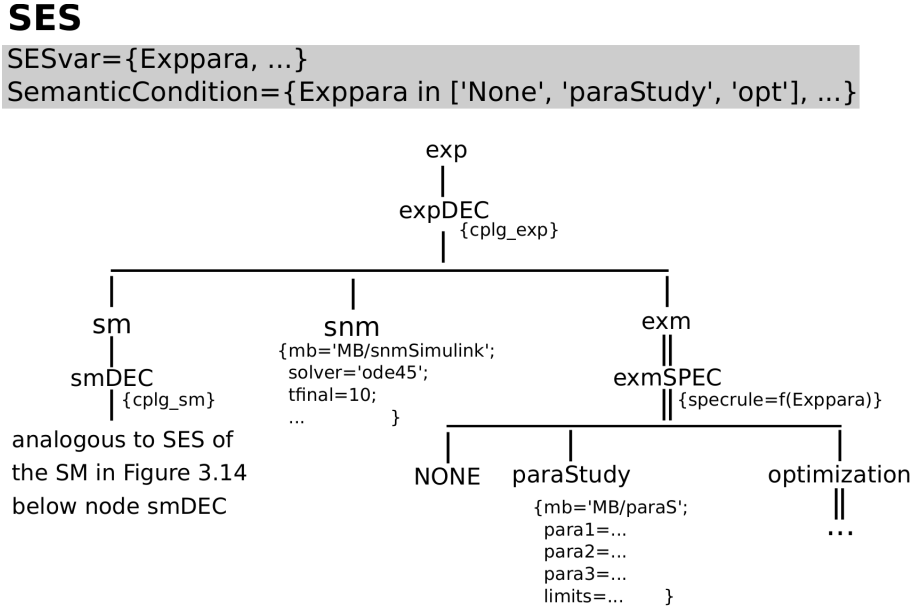


Figure 3.16: Principle SES with an experiment specification and the feedback control system with EF as SM (see Figure 3.14).

configuration as overall result,

- else return that the goals cannot not be reached with these model structures and parameter variations.

Figure 3.17 shows result plots of the described experiment using a step disturbance and two parameter variations for the PID controller.

A detailed software architecture for the description and execution of highly complex experiments based on the SES/MB framework is presented in Chapter 4. In extension to the work of Schmidt [135], it is investigated how a simulator-independent architecture based on *Functional Mock-up Interface* (FMI) can be realized.

3.3.4 Challenge for Experiments with the SES/MB Framework with Multiple Simulators

The modular-hierarchical design of the EF and the SnM as well as the ExM allows the integration of all components into the SES/MB framework. Schmidt [135] describes the EF, the SnM, and the ExM as components that can be organized in an MB. Thus, all parts of a simulation experiment can be described in an MB. The MB usually organizes simulator-specific components. The goal in this work is to show the experimentation on multiple simulation tools.

The components of the EF are part of the SM and thus these are executed during a simulation run. The EF configures the MUS and controls a simulation run. In doing so, a simulation run can be aborted *immediately* if limit values given for the simulation are violated. For this purpose, the live simulation data of the MUS must be accessed. This means that the components of the EF are simulator-specific, just like the MUS.

Different simulation tools also have different interfaces for simulator configuration. Thus

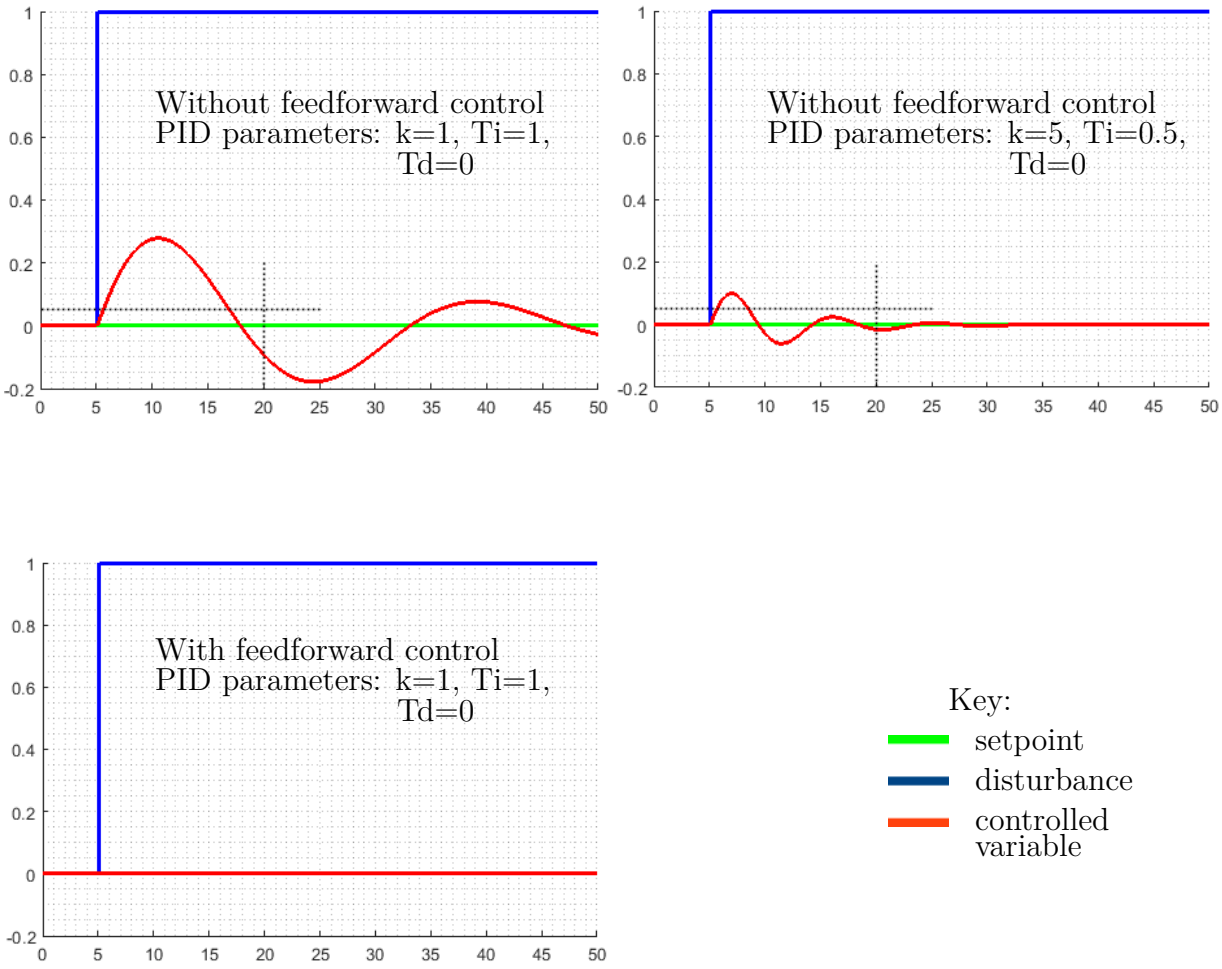


Figure 3.17: Result plots of the experiment with two parameter variations and two model structures.

the specification of the component SnM is identified as the main challenge in the experiment structure. One part of the SnM describes the configuration of the simulator. Therefore an interface to the simulation tool is needed. This part of the SnM specifies simulator-independent properties such as t_{start} or t_{final} as well as simulator-specific properties such as the simulation method of the simulator, i.e. the *ODE solver*. A simulation tool often provides multiple solvers. Each solver differs in the name and the parameterization. According to Schmidt [135] the SnM can be adapted by setting its method specific parameters, called P_{SnM}. This allows the setting of different configurations for one simulation tool. Apart from different solvers in one simulation tool, in multiple simulation tools solvers with the same algorithm usually have different names and parameters. Different solvers in one simulation tool have the same configuration interface, the interface in other tools may differ. For different simulation tools the interface of the SnM needs to be specific. Furthermore the SnM controls the model execution, e.g. several simulation runs, as well as data acquisition. These tasks require an interface to the simulator as well. In different simulation tools this interface differs. The SnM therefore can not be organized as a simulator-specific or general simulator-independent component in the MB. Other

tasks of the SnM like the analysis of collected data or the data evaluation can be specified simulator-independently.

3.4 Summary

The discussed extensions of the SES/MB framework offer new modeling approaches, remove limitations, and support the integration with simulation-based experiments as well as the integration into software architectures. In particular, the introduction of SESvars and SESfcns makes modeling with SES more flexible. Variable coupling relations and selection rules can be specified much more easily and clearly. A further flexibilization follows from the mb-attribute for the definition of links to basic models in the MB. The node names and names of the basic models are completely decoupled and links to different MBs can be defined. Furthermore, the SESvars and semantic conditions can be used to define an interface of the SES for integration into software architectures. In this connection stands also the again revised pruning operation. With the introduced extensions for the pruning of hierarchical multi-aspects as well as the extensions for the pruning of combinations of descriptive nodes on a hierarchy level, so far existing restrictions of the automated pruning of an SES were removed.

The picked up concept of the EF for the separation of MUS from the context of the model application shows advantages and disadvantages in the opinion of the author. The EF allows a clear decoupling of MUS and application context. Thus, the EF clearly increases the clarity of modeling and the reusability of the MUS for complex models. On the other hand, an increased modeling effort is associated with the EF. For relatively simple models, such as the example model considered, the advantages and disadvantages should always be weighed.

The modular structure of simulation-based experiments and the clear separation between experiment methods and experiment control support a specification of experiments in analogy to FoMs. Basic experiment methods can be organized like basic models in an MB. The structure of experiments and variations of experiment parameters can be specified in the SES. By means of the pruning operation, not only an SM consisting of MUS and EF can be derived as a candidate, but also a simulation-based experiment. In the minimum case a simulation-based experiment consists only of an SM and an SnM, which interfaces the executing simulator with the SM and the EC. The SnM is needed in every simulation-based experiment for the execution of simulation runs. It is simulator-specific and implements a kind of wrapper for a concrete simulator. Complex experiments additionally include an ExM. The ExM is a numerical procedure without direct reference to an SM or simulator and can itself be composed in a modular-hierarchical way. The ExM communicates with the EC, initiates simulation runs via the SnM, and evaluates simulation results.

In summary, basic models, EF components, simulator-specific SnMs, and basic experiment methods for building ExMs can be organized in an MB. Structures and configurations of simulation-based experiments can be specified in an SES. By means of pruning, the structure and configuration of a candidate simulation-based experiment can be derived in the form of a PES or an FPES. If an appropriate build method is available, an executable simulation-based experiment can be generated from the PES or FPES using the MB. The execution of the experiment or the linking of simulation-based experiments is the

responsibility of the EC.

Critically it should be noted, that the basic models, the EF components, and the SnM are simulator-specific. Accordingly, the build operation is also simulator-specific. In Chapter 2 the generation of models or simulation-based experiments for investigations with different simulators was demanded. In order to support the build operation with different simulators, the basic models, the EF components, and the SnM need to be adapted during model or experiment generation. Consequently, a software component implementing the build method must contain simulator-specific code to a high degree.

The objective of the next chapter is to develop an SES/MB-based software architecture that supports both simulator-specific MBs and simulator-independent MBs and associated model generation methods.

4 An Architecture for Automating Simulation Experiments with Different Simulators Based on the SES/MB Approach

In the previous chapter, extensions of the *System Entity Structure* (SES)/*Model Base* (MB) approach were described. On the one hand, limitations in the modeling of *Families of Models* (FoMs) are removed and on the other hand, the integration of the SES/MB approach into software architectures is supported. Furthermore, concepts for the specification and generation of simulation-based experiments for FoMs were discussed.

In this chapter, a software architecture and procedures for the automation of simulation-based experiments based on the SES/MB framework are presented. A major focus is the specification and generation of simulation experiments using different target simulators. First, the motivation for the automation of simulation experiments and the support of different target simulators is clarified. Then, an existing architecture for performing automated SES/MB-based simulation experiments is analyzed and ideas for supporting multiple target simulators are discussed. Based on the previous analysis and discussion, a Python-based architecture that supports different target simulators is then presented. The individual components of the architecture and the challenges to be overcome are presented in detail. Finally, a short excursus on the specification and generation of non-simulation specific applications with the software architecture is given and essential aspects of the chapter are summarized.

4.1 Objectives and Demarcation

In engineering, a large number of different system configurations often has to be investigated by simulation, for example to determine the best possible design or to realize individually adapted system designs. In this thesis, a minimal case study of a control system is considered as an example. The aim is to find a minimal structure with the corresponding control parameters for defined targets. As a procedure model for simulation studies, the life cycle of a FoM for simulation was already considered in Section 2.1. The classic SES/MB framework offers a formalized approach to specify different system configurations and to generate an executable simulation model by means of the operations *pruning* and *build*. This covers steps 3 to 6 of the life cycle model according to Figure 2.1. However, the classic SES/MB framework does not support automation of simulation experiments and

simulation studies based on defined targets.

In real world applications, the execution of the necessary simulation experiments is often time-consuming and manual handling of steps 6 to 8 of the life cycle model is error-prone. Even a single simulation run often has a long runtime. Therefore, as few simulation runs as possible should have to be performed in order to obtain valid results. The planning and design of simulation studies has been the subject of years of research and a variety of workflows and frameworks have been developed. A comprehensive recent research on this was done by Schmidt in [135]. At this point, the author refers to the recent research by Gil et al. [58], by Teran-Somohano et al. [143], and by Ruscheinski et al. [129] as representative examples.

The focus of this work is the development and implementation of an architecture for the automation of simulation studies based on defined targets. The starting point for this is the architecture developed on the basis of the SES/MB framework according to Schmidt [135]. This architecture has been successfully used in projects of the *Computational Engineering and Automation* (CEA) research group at the Hochschule Wismar, University of Applied Sciences, such as in the common project with the University of Bremen which was supported by the *Deutsche Forschungsgemeinschaft* (DFG): *Model-based Planning of Energy-Efficient Process Chains in Machining Component Production with System Entity Structures (Modellbasierte Planung energieeffizienter Prozessketten in der spanenden Bauteilfertigung mit System Entity Structures, GZ: BR825/62-2 PA631/2-2, PA631/2-3).* However, Schmidt's [135] architecture assumes that only one specific target simulator is integrated and that it is connected to a compatible environment with numerical experiment methods. This condition is met in his choice of MATLAB/Simulink to implement a prototype.

Building on Schmidt, according to Chapter 1, the focal points of the architecture to be developed in this work are:

- the integration of various target simulators,
- the organization of MBs with models usable by different target simulators,
- the removal of existing restrictions on the pruning of SES with multiple aspects,
- improved tool support for SES modeling through online error checks, and
- the use of standards to support the exchange of SES models with other software systems.

Topic 3 has already been discussed in Chapter 3 and topic 4 and 5 are self-explanatory in terms of motivation. The motivation for topic 1 and 2 is also obvious. In the following, the motivation of topic 1 and 2 is discussed under the aspect of validation in *Modeling and Simulation* (M&S). As discussed in Section 2.4.1 operational validity is supported by the architecture as well.

Since the development of the architecture focuses on the integration of different simulators and the organization of a generic MB, aspects of the generation of simulation models for different simulators and the execution of simulation runs are dealt with superficially, while their integration with other numerical methods is in the background.

4.2 Analysis of an Architecture for Automating Simulation Experiments

An architecture for automation of simulation-based experiments based on the SES/MB framework is introduced in Pawletta et al. [111] and Schmidt et al. [137] and further developed by Schmidt in [135]. The architecture complements the classic SES/MB framework with the following components and methods:

- an *Experiment Control* (EC) for the specification of experiment goals and steps of an experiment process,
- a combined MB&*Experiment Method Base* (EMB) that is a repository with basic system models and experiment methods, such as optimization methods like Nelder/Mead etc.,
- a pruning method for automated processing of an SES to derive an admissible simulation-based experiment configuration,
- a build method for generating an *Executable Simulation-based Experiment* (ESE), and
- an *Execution Unit* (EU) for the execution of generated ESEs.

According to Schmidt, an *experiment process*, in short an *experiment*, can consist of several simulation-based experiments, defines necessary parameters, evaluates results of single simulation-based experiments, and determines overall results of the entire experiment. The complemented architecture is presented in Figure 4.1.

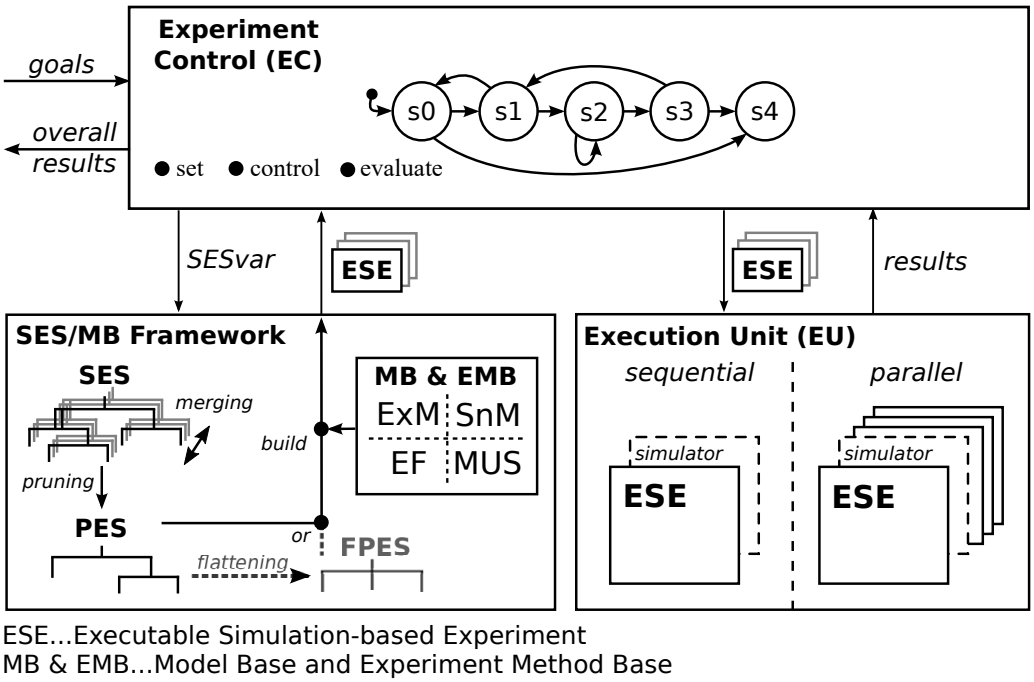


Figure 4.1: SES/MB&EMB-based architecture for automating a set of simulation-based experiments developed in style of Schmidt [135].

In the *modeling phase* a set of alternative system configurations is specified in an SES. A

system configuration representing a simulation-based experiment according to Section 3.3.3 (Figure 3.15) comprises of an *Experiment Method* (ExM), a *Simulation Method* (SnM), and the *Simulation Model* (SM) with *Experimental Frame* (EF) and *Model Under Study* (MUS). Note, that a system configuration represents a simulation-based experiment and that the SES/MB framework has to generate an ESE. For this purpose, the MB is extended to an MB&EMB. That means, in the modeling phase all necessary basic system models, including EF components and the different ExMs have to be organized in the MB&EMB. In addition, the MB&EMB has to contain an SnM. Furthermore, the experiment goals and the experiment process need to be defined in the EC unit. The EC can be specified in different ways, for example as a simple control script or as a finite state machine.

The EU is a kind of wrapper for the target simulator and its experiment environment used in the architecture. The wrapper provides an *Application Programming Interface* (API) that can be used by the EC. Schmidt [135] assumes that only one type of target simulator is integrated into the architecture. Moreover, it is integrated into an environment with experiment methods, such as numerical optimization methods.

In the *execution phase*, a model and experiment configuration, which represents one simulation-based experiment, is derived from the set of configurations specified in an SES using automated pruning of the SES. For the automated derivation of a configuration, *SES variable*s (SESvars), which are input variables of the SES/MB framework, are assigned values in the EC. The values can also be assigned reactively on the basis of experiments that have already been carried out. The result is a *Pruned Entity Structure* (PES) that can be flattened to a *Flattened Pruned Entity Structure* (FPES) as described in Chapter 3. It should be emphasized again that the PES or FPES both describe a model configuration and its integration into a simulation-based experiment. Based on the configuration information in the PES or FPES, the build method generates an ESE using basic system models, EF components, an ExM, and the SnM organized in the MB&EMB. The EC receives the generated ESE and can add or update information to the execution of the ESE. Such information can be, for example, data for the SnM, such as the duration of a simulation run, or data for the ExM, such as the termination accuracy in a numerical optimization. The full configured ESE is sent to the EU, executed under its control in a sequential or parallel fashion and after completion the experiment results are returned to the EC. The EC then analyzes the results. Thus, the derivation and generation of subsequent simulation-based experiment configurations can be controlled reactively based on simulation-based experiments already carried out. The necessity for reactivity is already shown by the control example introduced in Section 2.5.1. The simulator explicitly shown in the EU in Figure 4.1, again emphasizes that the SnM, as shown in Figure 3.15, is only an interface component and not the target simulator itself.

In Schmidt's [135] architecture, a generated ESE is always specific to a target simulator with integrated experiment execution environment. Schmidt provides the proof of concept with a prototype implementation for the MATLAB/Simulink environment. However, Schmidt [135] does not investigate a simulator-independent description and execution of simulation-based experiments. The SES represents a nearly simulator-independent specification of experiment configurations, but the basic models and the experiment methods organized in the MB&EMB are simulator dependent. The component EC can be specified in any language or notation independent of a specific target simulator. According to Figure 4.1, the SES/MB framework and the EU must provide well-defined interfaces

(APIs) that are used by the EC.

The SES/MB framework includes the data structure of SES as well as the data structures of PES and FPES based on it, the model and experiment methods repository MB&EMB, and the methods pruning, flattening, merge, and build. The latter form the API of the framework. According to the discussion in Sections 2.5.3 and 3.1, the SES represents a simulator-independent specification of experiment configurations. In contrast, according to Sections 2.5.3 and 3.2.2, simulator-specific basic models are organized in the MB. The same applies to experiment methods in the EMB.

The implementation of a completely simulator-independent SES is often difficult in practice. If parameterizations for basic models in the MB are specified at the leaf nodes of the SES, these parameterizations must be compatible with the adjustable parameters of the basic models or explicitly adapted by the build method. An analogous problem can occur when specifying coupling relationships, since simulators use different input/output interfaces for basic models, such as port names or port types.

The methods *merge*, *pruning*, and *flattening* are simulator-independent. They work with the data structure of the SES and the derived data structures of the PES or FPES. The build method, on the other hand, which creates an executable ESE using the PES or FPES and the simulator-specific components of the MB&EMB, must be implemented simulator-specifically.

As mentioned above, the EU represents a wrapper for a target simulator including the associated experiment environment of the target simulator. It provides the EC with a defined interface for executing simulation-based experiments on a target platform using simulator-specific interfaces of the target platform.

4.3 Approaches to Support Multiple Simulators

According to Section 4.2, the build method represents the real challenge in supporting multiple simulators. The build method represents the union of the challenging components mentioned in Section 3.3.4. The basic models in an MB have so far been considered as simulator native components. Consequently, these components also have a simulator-specific interface using an appropriate syntax and semantics, such as:

- (i) different port names and port types,
- (ii) different parameters and names of parameters, and
- (iii) different parameter meaning.

To support different simulators, all basic models in an MB including the components of the EF and the SnM must have corresponding compatible interfaces.

Consequently, differences of the individual simulators must be resolved both in port names and port types and different parameter meanings must be aligned. As a result, several simulator-specific MBs are required for different simulators *so far*. Simulator-specific SnMs are organized in the MBs to account for the configuration interfaces of the particular simulator. This is a crucial difference to Schmidt's [135] approach, where only one SnM needs to be defined since only one simulator is supported. The build method

must support the different simulator-specific MBs, which means a specific model builder for each target simulator. This approach is called *native approach* in the following.

An alternative approach is offered by the concept of model exchange using *Functional Mock-up Interface* (FMI) according to Section 2.4.3. It organizes simulator-independent components in the form of *Functional Mock-up Unit*s (FMUs) in an MB. A build method generates simulation models as simulator-independent FMUs. Because each simulator is configured using its own interfaces, the SnM must still be implemented in a simulator-specific manner. This approach is called *FMI-based approach* in the following.

In the following, only simulator-independent model generation and execution in terms of *simple simulation-based experiments* according to Section 3.3.3 are considered. The generation of executable complex simulation-based experiments based on ExMs organized in an MB according to Schmidt [135] is not considered. The approach requires that the M&S environment supports immediate integration of an SnM with various numerical ExMs. This requirement is not met by a large number of M&S environments. The framework presented in Section 4.4 takes an alternative approach.

In the next two subsections, different approaches to organize the MB as well as to implement the build method are discussed for the native approach as well as for the FMI-based approach.

4.3.1 Native Approach

The native approach requires that a separate MB is created for each supported simulator and simulator-specific model builders are created. With the native approach, different conventions of the simulators regarding ports can be solved by embedding basic models in subsystems, if these support a configuration of ports. The subsystems are organized as basic systems in the native MB. Different approaches for adapting different parameterizations and interpretations of parameters are presented below.

Preconfigured MB The simplest approach is to organize all basic models preconfigured in a simulator-specific MB. This approach often leads to many basic models in the MB. Hence, it will be hard and costly to maintain. Sometimes simulators need specific basic models, such as special blocks for output variables.

Parameter Adaptation in the Model Builder This approach takes advantage of the fact that an individual model builder must be created for each simulator. In the model builders, simulator-specific adaptations of basic models can be made, e.g. setting of the correct parameters and adaptation of its syntax etc. A disadvantage is that much simulator-specific code must be implemented in the model builder.

Extend the MB This approach is based on the extension of the MB by a special function. This function adapts settings and syntax of parameters to configure the basic models. Any additional blocks needed for a simulator can be added and configured. Moreover, the function can code coupling information for using the additional blocks during model building. This leads to less simulator-specific code in the model builder, but coding the special function for configuration is costly and error prone for complex models.

Disadvantages of the native approach are that both simulator-specific MBs and simulator-specific model builders are required. The maintenance of different MBs is error prone and costly. Also, the effort and error-prone nature of creating simulator-specific model builders should not be underestimated. To overcome these issues, an FMI-based approach is explored below.

4.3.2 Functional Mock-up Interface based Approach

The FMI-based approach, unlike the native approach, requires only one MB and one model builder. When using FMI, the most obvious approach is to organize FMUs in the MB. These can be exported from any simulation program with FMI support. The basic models in the MB thus have compatible interfaces and can be used with various simulators that support FMI. The configuration and linking of basic models to an overall model by the model builder according to the information in the PES or FPES can be done according to different approaches.

Using the Target Simulator The basic models are imported into the target simulator and configured according to the PES or FPES. In the next step, the coupling information is evaluated and an overall model is generated. The generated model is simulator-specific. This approach requires an extensive API of the target simulator, since the configuration of the basic models and their linkage is done in the target simulator after import. Since the configuration of the model is done using the target simulator, the requirement of a simulator-independent model builder is *not met*.

Using the Target Simulator with Intermediate Simulator In this approach, the individual FMUs (basic models) are configured before import into the target simulator. For this purpose, the FMUs are imported into a simulator with comprehensive API, the intermediate simulator, and configured in it according to the PES or FPES. After configuration, the individual basic models are exported as configured FMUs and organized in the MB. Accordingly, target simulators no longer need to support the configuration of FMUs. The disadvantage of this approach is that importing and exporting FMUs is computationally intensive. Therefore the aim of FMI technology is not to build libraries of small FMU components (Junghanns and Blochwitz [71]). The generation of a complete model by linking the configured basic models is still done in the target simulator. Thus, the requirement of a simulator-independent model builder is *not fulfilled*. One advantage of this approach is that the structure of the model is accurately represented in the target simulator.

Using an Intermediate Simulator and Export an Entire Model as FMU In this approach, FMUs are imported from the MB into an intermediate simulator with comprehensive API, configured in it according to the PES or FPES, and coupled to form an overall model. This overall model is then exported as a model as FMU. This *model FMU* can be run in any target simulator that supports FMI. The approach only requires a simulator-specific model builder for the intermediate simulator, which generates the model FMU for all target simulators. The disadvantage of the approach is that the structure of the model is not represented in the target simulator. It is a blackbox model. Thus, its structure cannot be changed and the single model components cannot be debugged easily.

However, single model components can be debugged, if the model FMU specifies ports to the model components. By connecting the ports to instruments in a simulation tool, the behavior of the individual model components in the model FMU can be captured.

Using the new FMI companion standard *System Structure & Parameterization* (SSP), the generated model FMU can consist of one or more FMUs including parameterizations. With the software architecture presented in Section 4.2, this is not necessary because different structural variants are derived by pruning the SES and customized models are generated.

The FMI-based approaches support model generation for multiple simulators using one MB. The first approach requires an extensive API of the target simulator. This requires extensive support of the target simulator by the model builder. In the second approach, the use of the target simulator's API is limited to the import of FMUs. In the third approach apart from the import of FMUs the API of the target simulator is not used and thus this approach fulfills the requirement of a cross-simulator model builder.

4.4 Architecture Supporting Various Simulators and its Implementation as Python Toolset

The section introduces the overall architecture and basics of a prototype implementation in Python. After that the components and their implementation in Python software tools are discussed in detail. Special attention is paid to the support of the *build* method for different simulators. Both, system classes and their support in different M&S tools are highlighted and tool-specific features and limitations are underlined. Differences in model execution are explained. Finally, the generation of non-simulator-specific applications is presented.

4.4.1 Overview of the Architecture

Based on the architecture for automating simulation experiments analyzed in Section 4.2 for a specific M&S environment, such as MATLAB/Simulink, an SES/MB-based architecture for simulator-independent specification, generation, and execution of simulation models is presented. Furthermore, its prototypical implementation as a Python toolset is discussed. Figure 4.2 shows the components of the architecture, the basic relations, and the assignment of software tools. Additionally, it shows the basic steps of M&S using the architecture. In Figure 4.3, the relations are detailed in the form of interactions. The architecture includes the three components: (i) SES/MB framework, (ii) Experiment Control (EC), and (iii) Execution Unit (EU). The components are prototypically implemented in the form of four Python tools: (i) SESToPy, (ii) SESMoPy, (iii) SESEcPy, and (iv) SESEuPy.

In the *modeling phase*, a FoM is first specified as SES according to Figure 4.2. The modeling of the SES can be done modularly. With the merge method several SES can be composed to one SES. Modeling and merging of SES is supported by the tool SESToPy (**S**ystem **E**ntity **S**tructure **To**ols in **Py**thon). In an MB, basic models are organized as simulator native and/or FMU components. In addition, the EC defines experiment goals

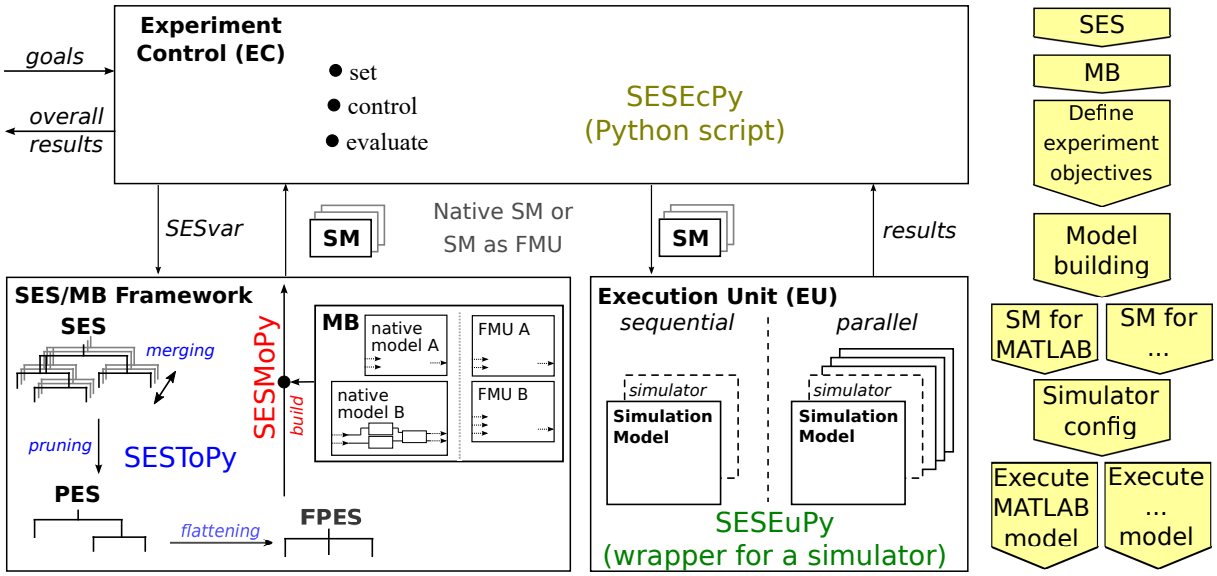


Figure 4.2: (left) SES/MB-based architecture for simulator-independent specification, generation, and execution of simulation models and their prototypical implementation as Python toolset; (right) basic steps for M&S using the architecture.

and experiment steps for conducting a *Simulation-based Experiment* (SBE). The EC is coded software-wise in the form of a script based on the template SESEcPy (**S**ystem **E**ntity **S**tructure **E**xperiment **c**ontrol in **Py**thon).

The basic steps in the *execution phase* of a simple SBE are: (i) model building, (ii) simulator configuration, and (iii) simulation model execution. The model building starts with the derivation of a concrete system configuration by pruning the SES on the basis of an assignment of the SESvars given by the EC. Optionally, flattening can be used to reduce the hierarchy depth. The methods pruning and flattening are supported by SESToPy using an API. The exchange format is an *Java Script Object Notation* (JSON) or *Extensible Markup Language* (XML) representation of the SES, as shown in an example in Appendix D. The next step is the build operation, which includes the generation of an SM based on the information coded in the PES or FPES and using components from the MB. The MB can include simulator native basic models and FMUs. The build operation supports native and FMI-based model building according to Section 4.3. According to Figure 4.3, a simulator-specific model can be a *Simulation Model Executable* (SME) or a *Simulation Model Representation* (SMR) depending on the target simulator. An SME can be executed directly by the target simulator, while an SMR contains instructions on how to create a model in the target simulator. The SMR is thus a kind of "makefile" for a simulator. The additional *config file* stores information of the model building, such as the assignment of SESvars, the target simulator, and the interface used, which can be native or FMI. The config file serves as interface specification for the model builder, EC, and EU components. The data exchange between the components is done via links. The complete build operation is implemented in the tool SESMoPy (**S**ystem **E**ntity **S**tructure **Mo**del builder in **Py**thon).

After model building, the EC adds experiment-specific information about the simulator configuration to the config file. These are for example the *Ordinary Differential Equation* (ODE) solver to be used and the simulation start time t_{start} as well as final time t_{final}.

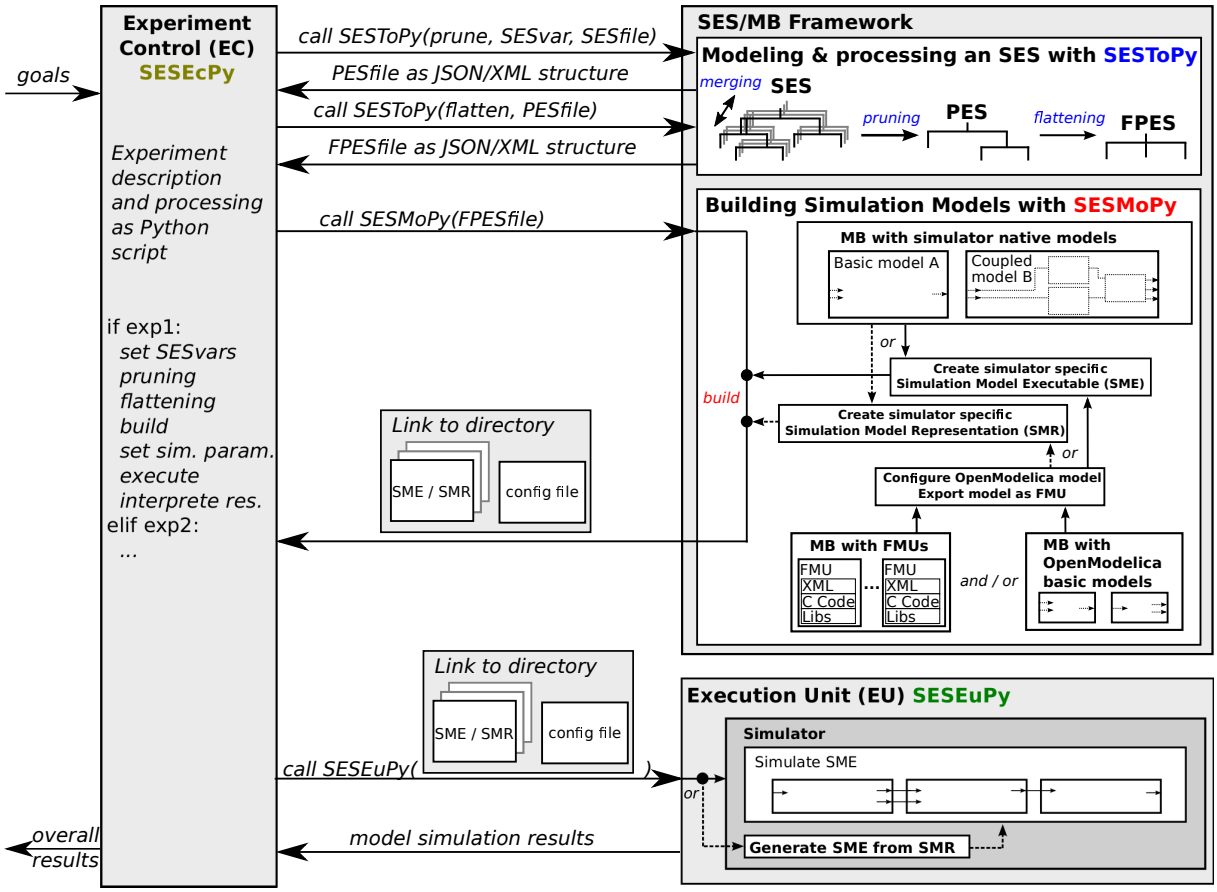


Figure 4.3: Basic interactions for generating and executing simulation models using the SES/MB-based architecture.

The SME/SMR and the config file are sent to the EU. The EU represents a wrapper for different target simulators. If the SM is available as SMR, the SMR is converted to an SME in the target simulator. Subsequently, the simulation is executed. Simulation results are read from the target simulator and returned to the EC. There, a decision about a new assignment of the SESvars and the start of another cycle can be made reactively or the experiment can be terminated. The EU is implemented in the tool SESEuPy (**S**ystem **E**ntity **S**tructure **E**xecution **u**nit in **Py**thon).

The basic M&S steps on the right side in Figure 4.2 show the process by the example of MATLAB/Simulink. The individual components of the architecture and their implementation in software tools are discussed in detail below.

4.4.2 Modeling and Processing an SES

The SES ontology has been comprehensively introduced in the previous sections. However, only few software tools exist for SES-based modeling, with none supporting the extensions developed as part of this work. Furthermore, SES tools as in Schmidt [135] or Zeigler and Sarjoughian [165] are based on specific M&S environments. A validation of the introduced architecture and the SES extensions requires a prototypical implementation. Therefore an appropriate tool, called SESToPy has been developed.

SESToPy is a modeling tool implementing a graphical SES editor and all related methods for processing an SES. Figure 4.4 shows a screenshot of a specific SESToPy view. In the editor an SES tree describing different system configurations can be specified interactively in a file browser view. For every node specific attributes or rules, such as couplings, specrules etc. can be defined according to the SES theory. Furthermore, global settings, such as SESvars, semantic conditions etc. can be defined. *SES functions* (SESfcns) have to be specified as Python functions and imported into SESToPy, which performs a comprehensive syntax check.

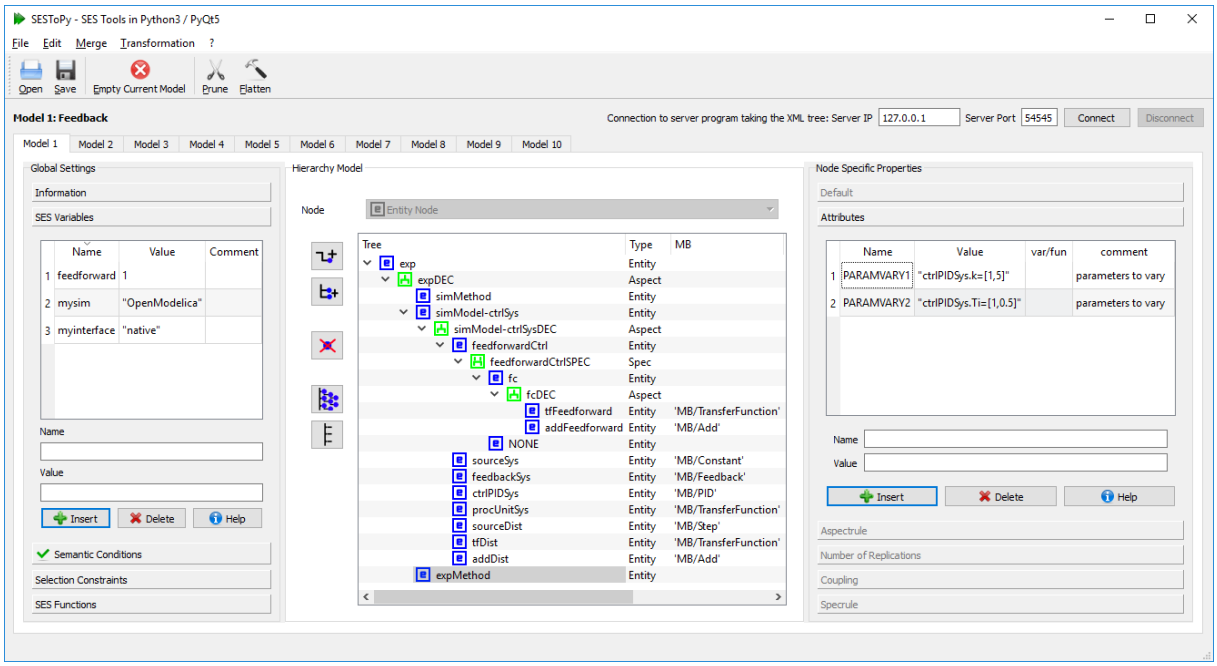


Figure 4.4: Screenshot of SESToPy with loaded SES of the feedback control system example extended by nodes specifying an SnM and parameters for performing a parameter study.

SESToPy supports all methods for processing an SES as introduced in Section 3.2.2, i.e. merging, pruning, and flattening. The pruning and flattening methods can be used interactively or as API methods. SESToPy supports the export of SES, PES, and FPES as JSON or XML files. The XML export is based on the schema mentioned in Section 3.2.1.

For supporting the modeling process an additional tool, called SESViewEl (**S**ystem **E**ntity **S**tructure **View** **El**ectron), has been implemented. SESViewEl runs in parallel mode to SESToPy and provides a second SES view, as shown in Figure 4.5. Changes in the SES using SESToPy are updated on the fly in the tree view of SESViewEl.

The screenshot of SESToPy in Figure 4.4 shows the SES of the previously introduced feedback control system example. This example does not include an EF. However, the SES includes the two nodes *simMethod* and *expMethod*, but they have no mb-attribute as discussed in Section 4.2 and 4.3. The ExM describes parameters for a simulation study. The SnM defines attributes, which specify the name of a concrete simulator, such as OpenModelica or Simulink, and its interface. The interface can be *native* or *FMI*. These attributes are configured using the SESvars *mysim* and *myinterface* as depicted in the left part of Figure 4.4. The SESvar *feedforward* is part of the SES in Figure 4.4.

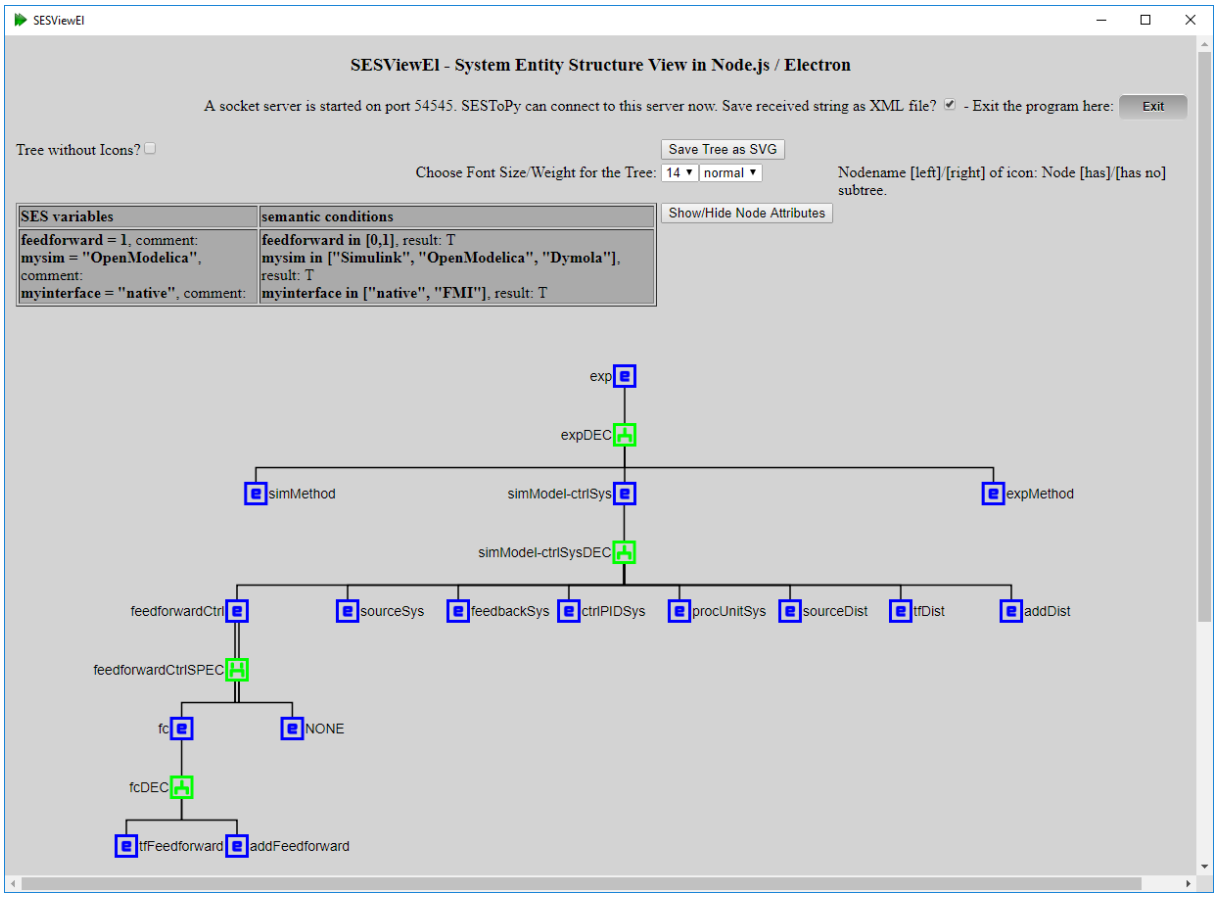


Figure 4.5: Screenshot of SESViewEl showing the corresponding tree view of the SES in Figure 4.4.

In contrast to the architecture of Schmidt [135], SnMs and ExMs are not organized in the MB. However, the SES may contain nodes for parameterizing SnMs and ExMs. The model builder must recognize these nodes using predefined keywords, such as SIMULATOR, INTERFACE, PARAMVARY, etc., and generate specific instructions for the configuration of the target simulator and integrated numerical experiment methods, respectively.

4.4.3 Organizing an MB

For model generation basic models need to be organized in an MB. Their interface must be compatible to supported target simulators regarding ports and configuration parameters. In Section 4.3 the native as well as the FMI approach are discussed in order to achieve a common interface. Thus, in this section organizing native MBs as well as an MB using the FMI approach are discussed. The creation of native MBs using the simulation tools Simulink [87], OpenModelica [103], and Dymola [38] as well as DEVSimPy [22] is demonstrated. Moreover, the generation of FMUs using OpenModelica is described as well. However, an FMU can be exported from any simulator supporting FMI. While simulator-specific MBs are one or several simulator-specific files, a simulator-independent MB is a directory containing FMUs. The organization of *native* MBs is discussed first.

For *OpenModelica* and *Dymola* a package in a specific file serves as MB. These simulation

tools build on the Modelica language and use the Modelica Standard Library as basic simulation library [91]. Modelica is a declarative modeling language especially designed for physical modeling. A Modelica model component is described by differential, algebraic, and discrete equations and it is structured in an object-oriented way. The Modelica Standard Library provides general definitions for constants, units, continuous and physical model components, and system utilities. Thus with Modelica signal flow-oriented as well as physical models can be described. Based on the Modelica Standard Library other components in source code can be defined and other libraries, e.g. commercial libraries supporting a special domain, can be added. In the MB basic models are organized in source code or components that subclass existing components and adapt the interface, so a Modelica model can be described simulator-independently in the SES.

For *MATLAB/Simulink* in a specific Simulink file basic models derived from the Simulink Standard Library are organized. Subsystems are created that enclose and unify the ports of the basic models in the Simulink MB. In addition a function can be added. This function is executed before a simulation run and enables the configuration of basic models in the MB. Thus, all parts of the interface of Simulink basic models can be customized in the MB and a Simulink model can be described simulator-independently in the SES.

For *DEVSimPy* introduced by Capocchi et al. [23] the MB can be set up with atomic or coupled DEVSimPy models. The models can be specified in Python language using a text editor and can be stored and structured in libraries. Complex simulation models are created manually by using drag and drop from model libraries. DEVSimPy acts as an user-friendly interface for collaborative M&S of DEVS systems implemented in the Python language. It provides a *Graphical User Interface* (GUI) for the PyDEVS (Bolduc and Vangheluwe [15]) and PyPDEVS (Tendeloo and Vangheluwe [142]) simulation kernels.

A native Modelica MB for the feedback control system is presented in Figure 3.6 in Section 3.2.3. An MB for MATLAB/Simulink is presented in Figure 4.6. This MB uses subsystems to abstract the ports of the basic models. A template of an additional function to enable the configuration of basic models according to the discussion in Section 4.3.1 is presented in Listing 4.1.

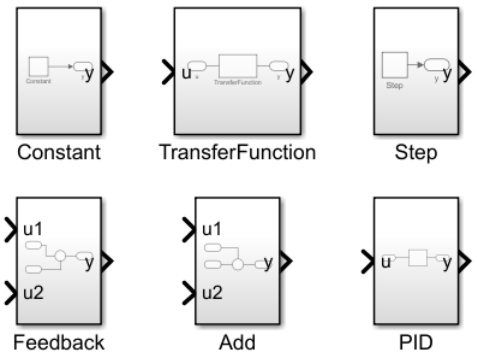


Figure 4.6: MB for the feedback control system with Simulink-based basic models.

```
%the current model
model = strtok(get_param(gcb, 'Parent'),'/');
%find all blocks
h = find_system(model, 'LookUnderMasks', 'on', 'SearchDepth', 2);
%go through all blocks
for k=1:length(h)
    if endsWith(h{k}, 'block name')
        %set parameters for block as defined in the SES
    elseif ...
        ...
    end
end
```

Listing 4.1: Template of an additional function to configure basic models for a Simulink MB.

In contrast to the native approach the FMI approach needs only one MB for various simulators built of FMUs. Therefore components can be exported from any simulator supporting FMI. OpenModelica e.g. has an interface to export FMUs for model exchange and for co-simulation. This interface can be used to export single or coupled components as FMUs manually and organize them in a directory forming a simulator-independent MB. Since OpenModelica is used as intermediate simulator, the MB can contain FMUs as well as Modelica basic models. An MB with Modelica basic models as well as FMUs is presented in Figure 4.7.

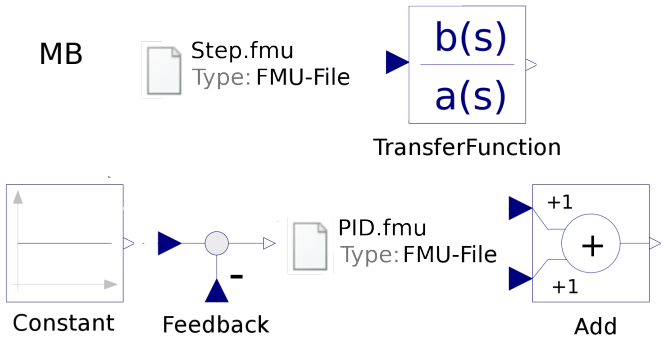


Figure 4.7: MB for the feedback control system with Modelica and FMI components.

4.4.4 Defining Experiment Goals and Steps in the Experiment Control

In the EC the experiment specific goals and experimentation steps are defined as a script using the template SESEcPy. Regardless of a specific experiment, calls to SESToPy, SESMoPy, and SESEuPy must always be executed by the EC. Therefore SESEcPy is structured in a general and an experiment specific part. SESEcPy assigns values to the SESvars based on the experiment description and starts the execution of the first SBE. This means SESToPy is called first for pruning with the current SESvar values and for flattening as well as SESMoPy for model building. The success of each step is returned to SESEcPy, otherwise the experiment is terminated. In the EU SESEuPy the execution is done returning concrete simulation results, which are evaluated in SESEcPy. Then, SESEcPy decides reactively how the experiment is continued by setting new values for the SESvars or SESEcPy terminates the experiment.

In the introduced feedback control system example with optional feedforward control the experiment specific part of SESEcPy includes the following steps:

- Try a system structure without feedforward control part.
  - Set the SESvar *feedforward=0* and simulate with different PID parameters, which are defined in the SES at node *expMethod*.
- If the experiment goals are reached with this system structure and one PID parameter set:
  - then return this structure and PID parameter set as overall result.
- Else try a system configuration with feedforward control part.
  - Set the SESvar *feedforward=1* and simulate with different PID parameters.
- If the experiment goals are reached with this system structure and one PID parameter set:
  - then return this structure and PID parameter set as overall result.
- Else:
  - return that the goals cannot be reached with these system structures and PID parameters.

In addition to the SESvar *feedforward*, the SESvars *mysim* and *myinterface* are set for configuration of the SnM for the next simulation. Additionally, simulator-specific information are added to the *config file* for the execution process as already mentioned in Section 4.4.1. These are information such as t_{start} and t_{final}, solver settings, sequential or parallel execution etc. An excerpt of a possible EC script for the feedback control system example is presented in Appendix E.

4.4.5 Model Building with SESMoPy

SESMoPy is a model builder, which supports the *build* method for several simulation environments in two ways: (i) *native model generation* and (ii) *model generation based on the FMI definition*. With the native model generation approach SESMoPy can build models for simulators supporting signal flow-oriented, physical, and discrete event models, whereas the latter type of models cannot be supported using the FMI approach. An overview and SESMoPy's model generation process are presented in an activity diagram in Appendix F. Essential aspects of the procedures for model generation are discussed in more detail in this section and for software generation in Section 4.4.7.

SESMoPy supports to build models of system classes using following simulators:

- signal flow-oriented systems using *Simulink*, *OpenModelica*, or *Dymola*,
- physical systems using *OpenModelica* or *Dymola*, and
- discrete event systems using *DEVSimPy*.

Subsequently, differences of these system classes are discussed in order to derive conclusions to implement model builders for the different simulators. In addition, the supported

simulators are briefly analyzed. Simulink with its blockset extensions and the Modelica-based tools OpenModelica and Dymola support the simulation of hybrid models, which are composed using components of different system classes. Moreover, they support the FMI standard. According to Elmqvist et al. [47] or Zupančič et al. [172] these tools are *general* M&S tools that can be used for different domains. Although OpenModelica and Dymola use nearly identical semantics, their APIs are different. DEVSimPy supports M&S of *Discrete Event System* (DES), based on *Discrete Event System Specification* (DEVS). Although DEVS forms a basis for all system classes (Vangheluwe [152]), DEVSimPy is focused on DES, but independent of any domain. SESMoPy is open to integrate other simulators such as Ptolemy II. *Domain-specific* tools regarding the characterization in Section 2.3.2 are not discussed in this work.

Signal Flow-oriented Systems Signal flow-oriented systems are represented by signal-flow graphs. The nodes represent units with inputs and outputs, which manipulate incoming signals and generate output signals (Mason [84]). The edges map signals between the units and have a direction of flow. In case of dynamic systems, a signal flow graph represents a set of linear and non-linear differential equations. This approach is classified as causal modeling. In M&S environments the nodes of signal-flow graphs are called blocks that have a certain behavior. The behavior of these blocks can be parameterized. Blocks define input and output ports. A model is built by coupling blocks. Output ports are connected with input ports.

Physical Systems Physical systems are described in an acausal form. The equations are stated in a neutral form without consideration of the computational order. Like signal flow-oriented systems physical systems can be modeled with blocks representing subsystems. The subsystems are connected by edges. Compared to signal flow-oriented systems the edges are not directed and one connection contains several variables of different type, i.e. potential variables and flow variables (Tiller [144]).

Discrete Event Systems Discrete event systems are characterized by events and states. State changes are caused by events, which can occur at any point in time in R_0^+. The set of system states over time is finite. There are different abstractions for modeling DES, such as the world views: event-oriented, activity-oriented, and transaction-oriented. Furthermore, there are state-based or network-based approaches, such as statecharts or Petri nets (Cassandras and Lafortune [26]). A general basis on which all approaches can be mapped is the DEVS theory according to Zeigler et al. [170]. DEVS theory strictly distinguishes between model specification and execution algorithms. Model specification is done with atomic systems, which represent dynamic behavior, and coupled systems. Coupled systems specify a set of subcomponents and coupling relations. Subcomponents can be of type atomic or coupled, allowing modular-hierarchical models to be composed.

OpenModelica and Dymola A core concept of all Modelica-based tools is the separation between the modeling language Modelica [91] and the algorithms for executing models. There are a number of M&S tools supporting Modelica, such as OpenModelica [103], Dymola [38], or SimulationX [48], and others. Due to the same underlying modeling language models created in one tool can easily be adapted for another tool. In this work the focus is put on OpenModelica and Dymola. The Modelica language is based on a Standard Library as well as various additional domain specific libraries. The libraries

organize parameterizable components with well defined port-based interfaces, which are physical or signal flow-oriented. Modelica supports hierarchical model structures.

Modelica models for OpenModelica or Dymola are coded in a textfile with the file name extension *mo*. This *textfile.mo* is called an SME, because it is directly processed by the OpenModelica or Dymola compiler. For the simulation the tools provide various configurable solvers. The compilation is performed in two phases. A Modelica compiler translates a model into a *General Purpose Language* (GPL) like C++, which is then translated by another compiler into a platform specific executable program. The tool controls the compilation and execution and collects *all* model variables, which can be depicted in a graphical way. The execution can be controlled by tool-dependent scripts. These scripts use tool-specific API commands, for instance to load a model, execute a simulation run with respective simulation parameters, and copy results in a file.

MATLAB/Simulink Simulink [87] is a signal flow-oriented M&S tool. Specific blockset extensions, such as Simscape or SimEvents, are not considered. Simulink is based on a Standard Library as well as various additional libraries. The libraries organize parameterizable components with ports that are identified by different numbers. Simulink supports hierarchical model structures. A Simulink model can be saved as SME or as SMR. For the simulation Simulink provides various configurable solvers, like OpenModelica or Dymola. The generation and execution of a Simulink model can be fully controlled using a MATLAB-based API. There are MATLAB commands for the specification, configuration, and coupling of Simulink blocks as well as for setting simulation parameters and the execution of a simulation. Using these commands, an SME can be generated from an SMR inclusive all simulation parameters. Thus, the entire execution process can be controlled by a MATLAB script. Unlike OpenModelica and Dymola, Simulink models usually specify the output of simulation results with special blocks in the model.

Of course, one could build an SME for Simulink as well using the Simulink modelfile format. The reason to build an SMR is practical. Since the SMR is a textfile, there is no need of an accessible Simulink in the model builder, but only when executing the model. This provides a clear separation between model generation and model execution as depicted in Figure 4.3. Hence, the build process can be executed on a regular computer, while the execution can be done on a more powerful computer or even on a *High Performance Cluster* (HPC).

DEVSimPy DEVSimPy is a DEVS-based discrete event M&S tool as introduced in Section 4.4.3. It provides direct support for processing SES/MB-based model specifications (Capocchi et al. [24]). DEVSimPy imports the XML structure of a PES or FPES and generates a DEVS-based model using basic models from the MB in Python source code. Then DEVSimPy links the model source code with its simulation kernels and translates the code to an executable *Platform-specific Model* (PSM). SESMoPy uses the model generation mechanisms of DEVSimPy. The simulation parameters for the executable can be set from the EC.

Model Building Using an FMI-based MB and OpenModelica as Intermediate Simulator In Section 4.3.2 approaches for using a general MB for various simulators based on FMI are discussed. SESMoPy builds on the *third approach* and uses OpenModelica as intermediate simulator to export an entire model as FMU that can be executed by different

simulators. Hence, the MB can organize basic models as FMUs as well as Modelica components. Figure 4.8 shows the basic steps for generating a model FMU. To simplify model building, the simulator independence of SES is restricted in that the specification of coupling relationships and parameter configurations for basic models must be in Modelica compliant syntax.

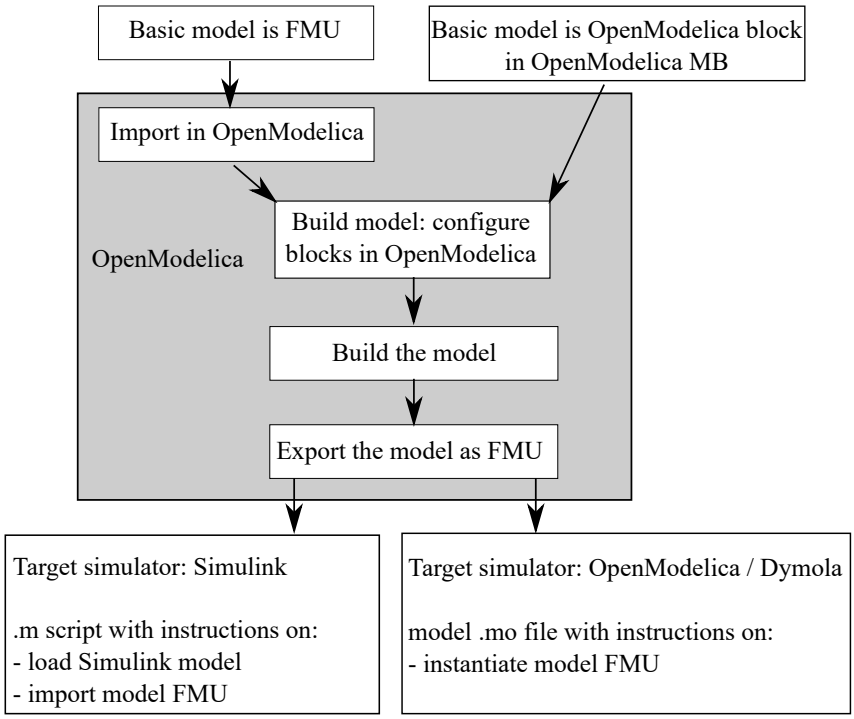


Figure 4.8: Basic steps for generating a model FMU and its preparation for a target simulator.

In case the basic model is an FMU, it is imported into OpenModelica and code is created as Modelica block interfacing functionality of the FMU. Afterwards the block is configured according to information in the FPES. Otherwise, if the basic model is a Modelica component, it is configured directly. For both cases finally a fully configured OpenModelica model is built using the couplings specification in the FPES. For usage of a model FMU by the Simulink simulator, the configured model FMU is automatically extended by output blocks, because Simulink does not return all model variables by default. This is depicted in Figure 4.9 by the example of the introduced feedback control system.

In OpenModelica output blocks depend on the type of the variable to be returned. The type of such variables can be determined from coupling information in the FPES. According to Section 3.2.1 the SES specifies port types in addition to port names. Figure 4.9 represents only signal flow-oriented couplings. Physical couplings are more complex, because physical ports comprise potential, flow, and stream variables, as shown by an example in Section 5.2. The fully configured OpenModelica model is exported as one *model FMU* with the added outputs as interface.

For FMI several tools were developed to support the development of software supporting FMI, some of which are introduced in Section 2.4.2. One tool is the “FMU Compliance Checker“ developed by the company Modelon AB [95]. SESMoPy makes use of this tool to check the model FMU exported from OpenModelica. The checked model FMU can be used in all simulators supporting FMI. The next steps of model building depend on the

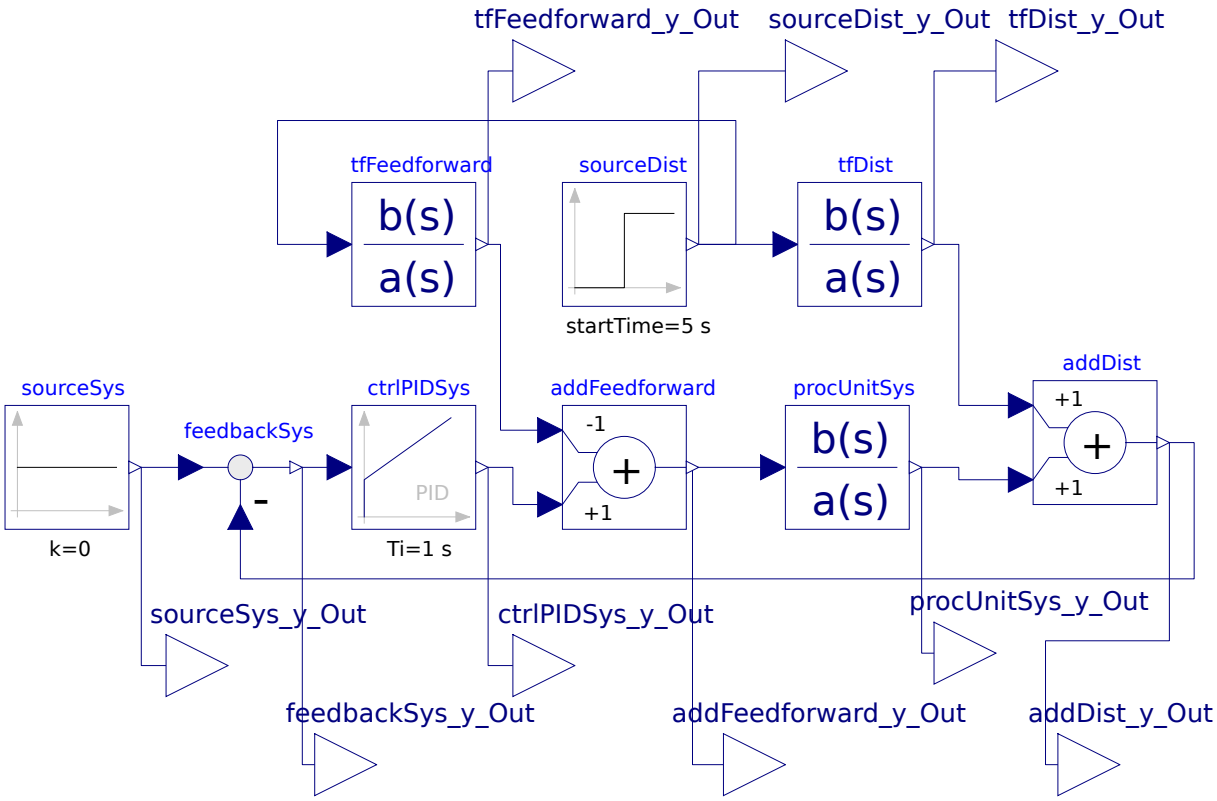


Figure 4.9: OpenModelica model extended with output blocks.

target simulator as depicted in Figure 4.8. The information about the target simulator and the kind of model building, native or FMI-based, is coded in the FPES at the node *simMethod*. In case of the target simulators OpenModelica and Dymola a *textfile.mo* is written with instructions to instantiate the FMU for execution in the target simulator. The FMU with the *textfile.mo* represents an SME according to Figure 4.3. In case of target simulator Simulink a *textfile.m*, called M-script, is written with instructions to instantiate the FMU in Simulink. The FMU with the *textfile.m* represents an SMR according to Figure 4.3.

The *Config File* To support several simulators the *config file* is introduced for the native and FMI-based model building approach. SESMoPy creates this file and stores information from the FPES specified at the nodes *simMethod* and *expMethod* as discussed in Section 4.4.2. Subsequently, this file is extended by the EC.

4.4.6 Executing a Simulation Model in SESEuPy

The EU SESEuPy is called by the EC SESEcPy. In SESEcPy the *config file* is extended with experiment specific information. In Section 4.4.4 this is specified as values, such as the solver algorithm, simulation start and final time (t_{start}, t_{final}) etc. Furthermore, it needs to be set which variables have to be evaluated for the experiment. For the feedback control system example these values are the setpoint, the disturbance, and the controlled variable.

For execution SESEuPy receives a link to the *config file* and a number of SMEs or SMRs.

The execution can be done sequentially or in parallel. For parallel execution SESEuPy creates a process for the simulation of each model and summarizes the simulation results on return of each process. On the executing machine it is now important that the target simulator is installed and its API methods are accessible. Depending on whether the model is an SME or SMR and thus on the target simulator the procedure differs.

For *OpenModelica* or *Dymola* as target simulator it is passed an SME that can be executed directly. SESEuPy generates a simulator-specific execution script with following commands:

- load the model,
- load the MB,
- execute the simulation according to the configuration passed in the *config file*, and
- get the simulation results and return them.

For *Simulink* as target simulator it is passed an SMR. Simulink returns simulation results using output blocks on model level. Under the requirement that the SES specifies simulator-independent model configurations, the output blocks are not modeled. In case of an FMU, the extension of a model with outputs for each component in SESMoPy as shown in Figure 4.9 is done to provide an interface for the FMU to generate. Variables of interest must be linked to Simulink output blocks for the native and FMI-based case. Therefore, SESEuPy extends the SMR by commands for generating output blocks and their couplings using the experiment specific information in the *config file*. In contrast to the target simulators OpenModelica and Dymola no separate execution script is needed for Simulink. Instead, the execution data is added to the SMR. Then SESEuPy calls MATLAB to generate an SME from the SMR according to Figure 4.3 and to execute the SME. In Appendix G an excerpt of an SMR and the generated SME is presented for the *native case*. Figure 4.10 shows an example of a generated SME for the *FMI case*.

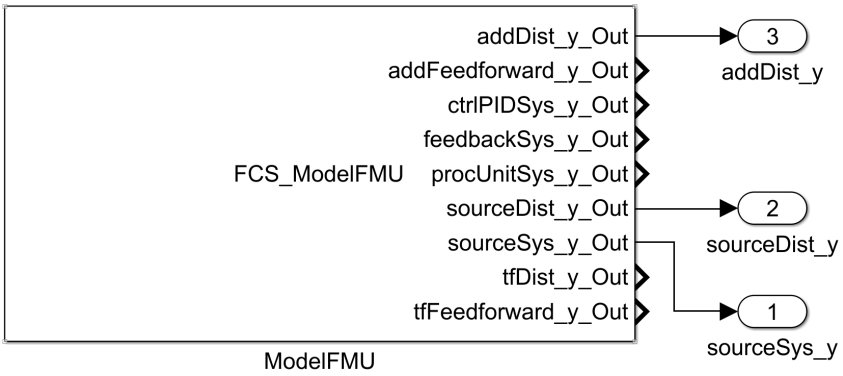


Figure 4.10: By SESEuPy generated SME for Simulink using FMI with added output blocks.

Finally, the EU SESEuPy collects simulator-specific the simulation results and sends them to the EC SESEcPy.

4.4.7 Generation of Non-Simulation Specific Applications

Due to the structure and well-defined interfaces the architecture and its prototype can also be used for modular-hierarchical component-based non-simulation specific applications. Subsequently, this is discussed for a component-based software system.

In the SES different variants of a software system are coded. Software components are organized in a repository that represents the MB. The mb-attributes at leaf nodes in the SES refer to these software components. In a function-oriented software system each component represents one function. Other attributes at leaf nodes can specify values for variables in the respective functions. Couplings then specify the structure of the software. Ports represent input or output parameters of functions. The node *simMethod* in the SES can be left away, because no simulator data has to be specified. The node *expMethod* can be used to specify different parameter configurations for the software system.

During pruning parts of the SES tree are cut away leaving only one structure. So the pruning and flattening processes do not differ from the derivation of simulation model variant. In the build process one directory for a software variant is created. In this directory software components are copied from the repository, which are linked by nodes in the FPES. Next, variables of the selected functions are set on source code level according to attributes specified at leaf nodes of the SES. During model generation, a coupled system consisting of basic systems is created on the basis of the FPES. In the same way, it is possible to create a main function during software generation. Using the coupling relationships, a sequence or hierarchy of function calls including the transfer parameters is then created. A small example for software generation is presented in Appendix I.

4.5 Summary

In this chapter, an architecture for performing cross-simulator automated studies of FoMs using the SES/MB approach has been discussed. For this purpose, an analysis of the architecture presented in Schmidt [135] is done first. In this architecture, the classical SES/MB framework is extended by additional software components and the MB by ExMs to form an MB/EMB. The MB/EMB is specific to a M&S environment. Not only simulation models of a FoMs, but simulation-based experiments can be generated. To execute these experiments, the M&S environment must provide an execution environment for the numerical ExMs in addition to the simulator. Furthermore, parameterizations as well as coupling-specific properties, such as port types, of basic models have to be mapped in the SES. This leads to a not completely simulator-independent SES.

A central challenge is the organization of a simulator-independent MB. The specification of simulator-specific aspects in the SES is omitted and SMs for different simulators can be generated. For the realization of a simulator-independent MB, a native and an FMI-based approach have been discussed. The native approach requires the extension of an MB by a function that implements simulator-specific adjustments during model generation. The specification of this function is costly and error-prone. Furthermore, the native approach requires simulator-specific model builders. With the FMI-based approach, no specific extensions to the MB are required. Exporting an entire model as FMU also achieves the requirement for a simulator-independent model builder. A disadvantage is that an FMU

is a blackbox model and thus cannot be debugged easily component by component.

In analogy to Schmidt [135], an SES/MB-based architecture for cross-simulator automated experimentation was designed and implemented as a Python toolset [51]. It supports interactive specification of SES and model generation for different target simulators using the native as well as the FMI-based approach. The EC and EU allow an automated execution and evaluation of simulation experiments. Since only a few target simulators support integration with numerical ExMs, such as MATLAB/Simulink, this aspect was not investigated.

Models can be assigned to different system classes. Simulators support different system classes. Based on the native approach, model builders for signal flow-oriented models were developed for the target simulators MATLAB/Simulink, OpenModelica, and Dymola. Furthermore, a model builder for DEVSimPy was created for discrete event-oriented models. Based on the FMI-based approach, model generation of signal flow-oriented and physical models was discussed using OpenModelica, Dymola, and MATLAB/Simulink as examples. The automated execution of simulation experiments was demonstrated using a signal flow-oriented model as an example.

Finally, the generation of a software system using the architecture was discussed. Analogies to the generation of simulation models were pointed out. The main difference is the absence of a simulation method. Thus the SES/MB approach is also suitable for the component-based generation of software systems.

5 Case Studies Solved Using the New Architecture

In the previous chapter the new *System Entity Structure* (SES)/*Model Base* (MB) architecture and its implementation in the form of a Python toolset were introduced and considered in terms of the signal flow-oriented feedback control system used in the thesis. In this chapter further examples of the application of the architecture are given with a focus on SES/MB-based modeling and native as well as *Functional Mock-up Interface* (FMI)-based generation of simulation models. Automation of simulation experiments and execution of simulation models are not considered. In the examples a *Discrete Event System Specification* (DEVS)-based system and a physical system are considered.

5.1 M&S of a Discrete Event System with Different Levels of Detail Using Native Model Generation

In this example a DEVS-based system with different levels of detail is examined. At first, modeling systems with different levels of detail is motivated. Ideas for modeling such systems based on the SES/MB approach are presented and a real-world case study is introduced. An SES describing system configurations for the system in this case study is developed, the associated MB is shown, a possible *Pruned Entity Structure* (PES) and its *Flattened Pruned Entity Structure* (FPES) are derived, and finally a model generated.

5.1.1 Multi-Resolution M&S

The permissible and reasonable level of detail of a model is determined by the objective of the investigation (Fishwick [49]). Models often need to be developed with different levels of detail. An example is the offline use of a model in the development phase and the online use of a model under real-time conditions in the operation phase. The execution of the models is often done with different simulators in the different usage phases. Models of this type are called multi-resolution systems according to Zeigler et al. [169]. The authors state that considering systems with different levels of detail is often beneficial:

- lower level of detail brings advantages in simulation speed and is easier to understand (limited parameters, consideration of groups of entities, ...) and
- higher detail enables more accurate simulation results and is valid in a higher scope of experimental frames.

Often, a simulation with a lower level of detail is sufficient to investigate a desired objective. Differences in detail need not apply to all model components of a system, such as shown by Santucci et al. [131] and Pawletta et al. [112].

In Chapter 2 a model was defined according to Zeigler et al. [170] as an abstract expression of objects of the real or imaginary world. Furthermore, it was said that a model always represents only a section of a larger world (Bossel [17], Zeigler et al. [167], Zeigler et al. [170], and many others). Depending on the application area and the study objectives, different degrees of abstraction with different forms of abstraction are used in modeling. According to Zeigler et al. [170] they are called simplification methods. In the following, the simplifications

- aggregation and omission as well as
- time abstraction

are discussed. Aggregation is the combining of system components in a single component and omission is the neglect of system components. Time abstraction is a special simplification in the modeling of dynamic systems. This includes, for example, with which temporal resolution dynamic behavior is abstracted, which Santucci et al. [131] refer to as time granularity.

According to Zeigler et al. [170], related models of different levels of detail from which candidate models are derived also form a *Family of Models* (FoM) as defined in Section 2.1. To describe the relationships among models with different details of abstraction, several authors such as Santucci et al. [131], Benjamin et al. [10], or Fishwick and Lee [50] propose a hierarchical modeling with different levels of abstraction. Hierarchical abstractions can be used to selectively hide elements that are not relevant to the objective of an investigation (Benjamin et al. [10]). However, according to Fishwick and Lee [50], it is important to have the ability to traverse levels of abstraction. Santucci et al. [131] propose an approach to model abstraction hierarchies based on the classic SES/MB framework. The proposal considers both, the composition of models with different levels of detail and the composition of models with different temporal granularity. For the specification of abstraction hierarchies, they extend the SES with two new descriptive nodes. A pruning operation is not specified for the extended SES. In the author's opinion, an automated pruning of the SES is made more difficult by introducing the new node types.

For modeling abstraction hierarchies, the general idea of Santucci et al. [131] was taken up and an SES/MB-based modeling approach was developed by the author without additional extensions to the SES (Folkerts et al. [52]). Thus, candidate models can be derived automatically using the regular pruning operation. The necessary interface and time resolution compatibility for the composition of models of different abstraction levels is solved via a *Downward Atomic Model* (DAM) and an *Upward Atomic Model* (UAM) per abstraction level. The DAM and UAM are application-specific basic models organized in the MB. In the SES, the DAM and UAM are referenced in attributes of ordinary entity nodes, and associated parameter settings are specified.

For comparison with the approach of Santucci et al. [131], a FoM of a watershed according to Santucci et al. [131] was specified with an SES (Folkerts et al. [52]). Executable simulation models were generated using the MB of Santucci et al., which was extended to include DAMs and UAMs. Thereby, the dynamic model components are organized in

a DEVS-based MB. The MB is created with the simulation software DEVSimPy. The investigations were performed with the architecture developed in Chapter 4. For the generation of DEVSimPy simulation models a simulator-specific *build* operation was implemented. The native interface must be used for modeling DEVS-based systems due to the limitations of FMI. In the following, the watershed problem is briefly introduced and the specification of a FoM with models of different level of detail with an SES is explained.

5.1.2 Introduction of the Case Study

The example concerns a watershed belonging to a mountainous part of France Alpes (Coron et al. [36]). It involves the rain and snow precipitations over a period of one year. Precipitation can be classified according to its nature into solid or liquid. Depending on the time of year and the geographical area concerned, the amount of solid or liquid precipitation will be different. The proportion moves between the two extremes: only snow or only rain. When precipitation falls as snow, the hydrological response of the watershed is not the same as observed in case of rain. There is a lack of response in the short term watershed due to the accumulation of solid precipitation in the form of a snow cover on the soil surface. When the conditions of melting are met, which can occur several days to several months after the occurrence of precipitation, this water is remobilized. It causes a delayed reaction of the watershed. The dynamic behavior of the watershed is mainly influenced by precipitation and temperature conditions. Usually, the precipitation data are day related and the temperature data are hour or day related. Based on these facts, the dynamic behavior of a watershed can be considered on the basis of different levels of detail. According to Santucci et al. [131] the level of detail can be considered regarding the decomposition of influences, called abstraction hierarchies, and the temporal granularity of data. They introduced the following levels of detail:

- In the simplest case, called watershed abstraction hierarchy level 0, the behavior is modeled just considering the percentage of rainfall on a daily basis.
- At the watershed abstraction hierarchy level 1, the behavior is expressed in more detail by taking into account the altitude and the flow associated with the three hydrogeological layers: (i) soil, (ii) surface and (iii) aquifer. At this hierarchy level, the altitude influence is broken down in two different levels of detail, called altitude abstraction hierarchy level 0 and level 1.
- Starting from the altitude abstraction hierarchy level 1, snow effects are considered on a daily or hourly basis, which are differentiated as basin time granularity level 0 and level 1.

From the above considerations follows a set of model variants with different levels of detail, which are specified using an SES.

5.1.3 M&S of a Watershed System

First, the different system configurations following from the levels of detail are modeled in an SES. Subsequently, a DEVSimPy MB of the watershed system and a possible model are shown.

Modeling of System Configurations Figure 5.1 shows an excerpt of the SES with the essential parts for describing abstraction hierarchies. The coupling relationships are not shown.

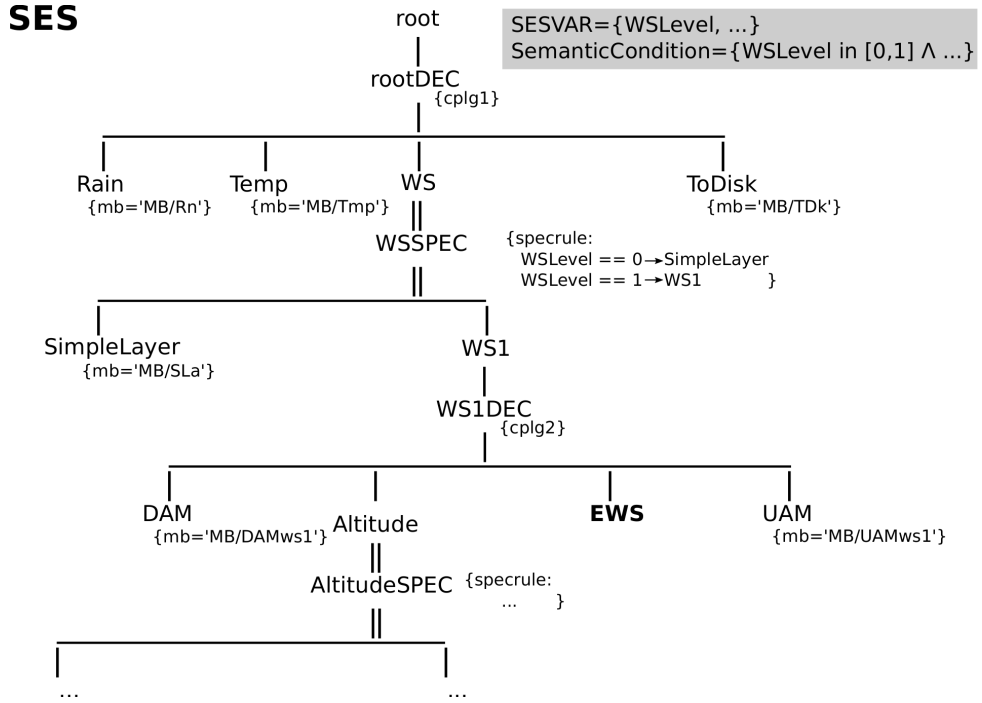


Figure 5.1: An excerpt of the SES for specifying model variants of the watershed with model components of different levels of detail.

As described in the feedback control example the aspect nodes describe system compositions, the specialization nodes variation points, and the leaf entity nodes links to basic models in the MB. The specification of the individual node attributes is analogous to the feedback control example. The variation points in the form of specialization nodes describe selection conditions with respect to their child nodes and thus the selection of submodels of different levels of detail including different time granularity. The third hierarchy level of the SES shows with the leaf nodes DAM and UAM the referencing of the above mentioned specific basic models for the adaptation of input and output interfaces and time resolutions.

The basic properties to consider when modeling a watershed are the amount of rain and the ambient temperature. Thus, the root node of the SES is composed of the leaf entity nodes *Rain* and *Temp*, as well as the inner entity node *WS* describing the watershed itself, and an auxiliary leaf entity node *ToDisk*. The composition is expressed with the aspect node *rootDEC*. The coupling relations are specified in the attribute *cplg1*. Each leaf node has an mb-attribute, which defines a reference to a basic model in the MB. The watershed *WS* can be either of the type *SimpleLayer* or of the type *WS1*. *SimpleLayer* constitutes the watershed abstraction hierarchy level 0, whereas *WS1* constitutes watershed abstraction hierarchy level 1. For selection during pruning, the watershed *WS* has a descriptive node of type specialization *WSSPEC* as child. Specrules defined as attribute of *WSSPEC* are evaluated. They allow the selection of one child node depending on the *SES variable* (SESvar) *WSLevel*, which has to be defined before pruning. *WS1* is composed of the four sibling nodes: *DAM*, *Altitude*, *EWS*, and *UAM*. The entity *WS1* represents a coupled model in the context of modular-hierarchical modeling. It specifies a more detailed version

of the *WS* as its sibling node *SimpleLayer*. To accommodate the finer time resolution and interfaces of this abstraction layer, this layer specifies references to corresponding interface models in the MB using the UAM and DAM nodes, which solve these problems.

The *EWS* node shown in bold is an entity node at which another SES defining the system configuration of an elementary watershed could be merged according to the full specification in Santucci et al. [131]. It can be considered as a leaf node for the purpose of this study. The fourth sibling of the composition of *WS1* describes the *Altitude* of the watershed. Depending on the level of detail the *Altitude* can be of different type. This part of the SES is not described here, since the modeling of the abstraction levels is done analogously as described before. Different temporal granularity can be treated like hierarchical abstraction from the point of view of the SES and its methods. However, the full SES and all couplings are presented in Appendix H.

DEVSimPy Watershed MB The MB organizes like in Santucci et al. [131] and Folkerts et al. [52] basic models according to the DEVS specification. For this example the *Watershed* library has been defined (Santucci et al. [131]). Figure 5.2 shows an excerpt of this library. Basic models are also shown, which are not referenced in the mb-attributes in the excerpt of the SES shown in Figure 5.1. In order to generate an executable model, the basic models have a simulator-specific interface. This means that simulator-specific model builders are required according to Section 4.3.1.

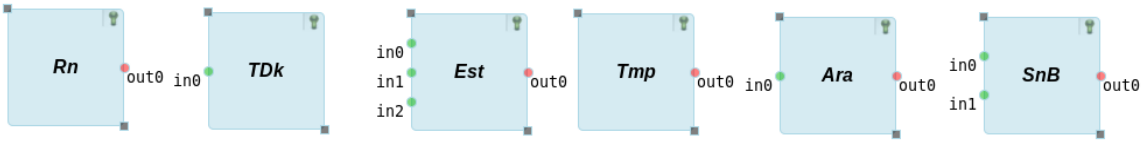


Figure 5.2: Excerpt of the MB in DEVSimPy.

Derivation of a Possible PES and Generation of an Appropriate Simulation Model The pruning method is used to select an abstraction variant and thus a model variant. For this purpose, values are assigned to the SESvars and a selection for a model variant is made according to the specrules. Couplings are adapted as described for the feedback control system example in the previous chapters. A section of a possible PES is shown in Figure 5.3. This PES represents the variant with a higher level of detail in *WS* and *Altitude*.

The PES can be flattened for easier application of the *build* method. The resulting FPES and the couplings for the highest level of detail of the watershed system are depicted in Figure 5.4. The corresponding model is shown in Figure 5.5 with the component *EWS* representing a coupled system. The build method is not discussed in detail, because it corresponds to the presentation in Chapter 4. However, the FPES in Figure 5.4 includes nodes that are not shown in the SES and PES in Figures 5.1 and 5.3. Different PES, the corresponding FPES, and generated models with different levels of detail are presented in Appendix H.

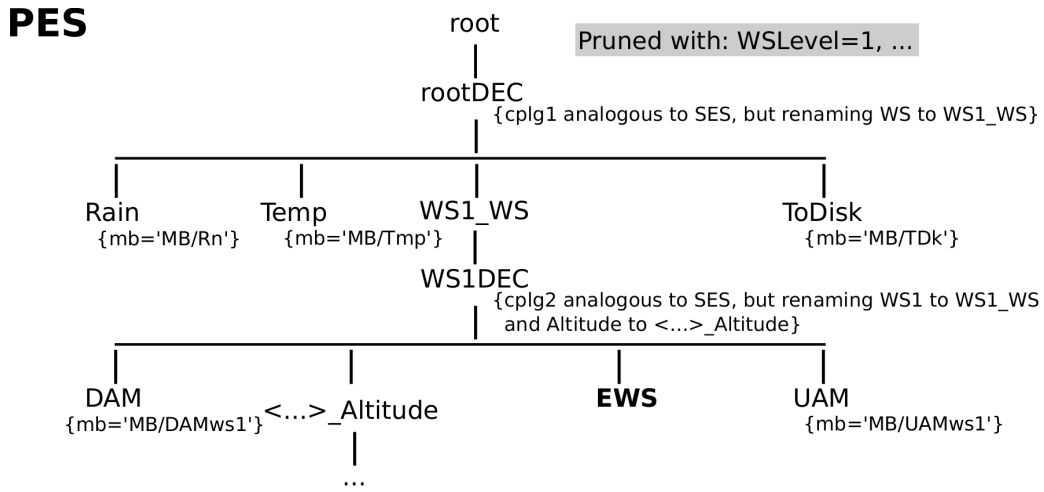


Figure 5.3: An excerpt of a possible PES of the watershed.

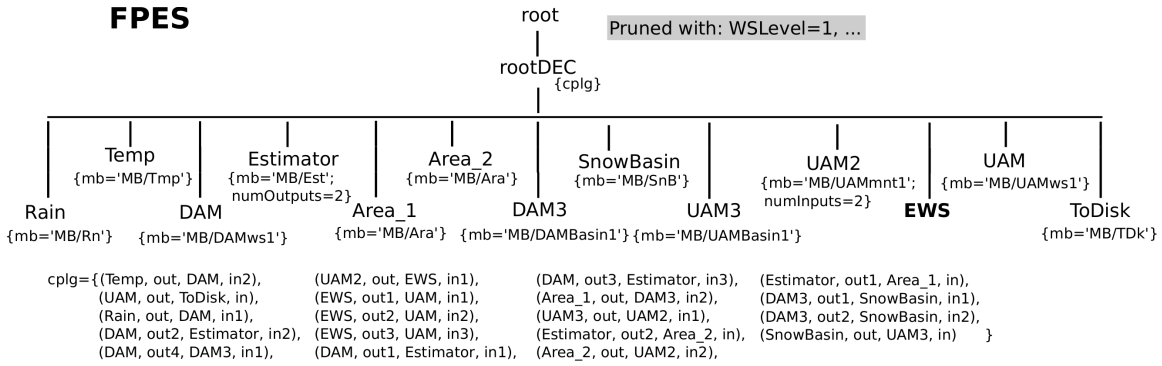


Figure 5.4: A possible FPES of the watershed representing the highest level of detail.

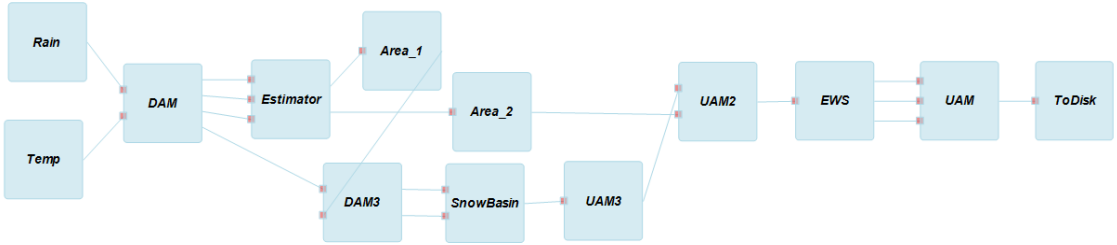


Figure 5.5: The model built with the FPES in Figure 5.4 and the MB in Figure 5.2.

5.2 Feedback Control System with Physical System Using Functional Mock-up Interface based Model Generation

This example builds on the FMI-based model generation introduced in Chapter 4, showing the generation of physical models using the SES/MB approach in conjunction with FMI. In the first section, the introduced feedback control system is extended by a physical process and the integration into simulation experiments is discussed. The differences of physical models to signal flow-oriented models will be discussed. Starting from the already introduced SES, the modifications are discussed, the MB is introduced, and the derivation of a possible FPES is shown. Furthermore, the coupling relations and the *build* method with an FMI-based model builder are explicitly discussed.

5.2.1 Modeling of a Physical Process for the Feedback Control System Example and the Integration into an Experimental Frame for Simulation Experiments

As introduced in Section 2.3, *Cyber-Physical Systems* (CPS) combine both physical components and software components. CPS are often modeled with the Modelica language (Buffoni et al. [21]). A physical component can, for example, consist of electronic and mechanical subcomponents. These have a transfer behavior and can be described with transfer functions or physical models. In the example of the feedback control system, the real process to be controlled can be modeled as a physical component as an alternative to the transfer function. Similarly, disturbances can also be modeled physically. Accordingly, in the models of the feedback control system in Figure 3.7, the components *procUnitSys*, *sourceDist*, *tfDist*, and *addDist* can be replaced by a physical system. This is shown in Figure 5.6.

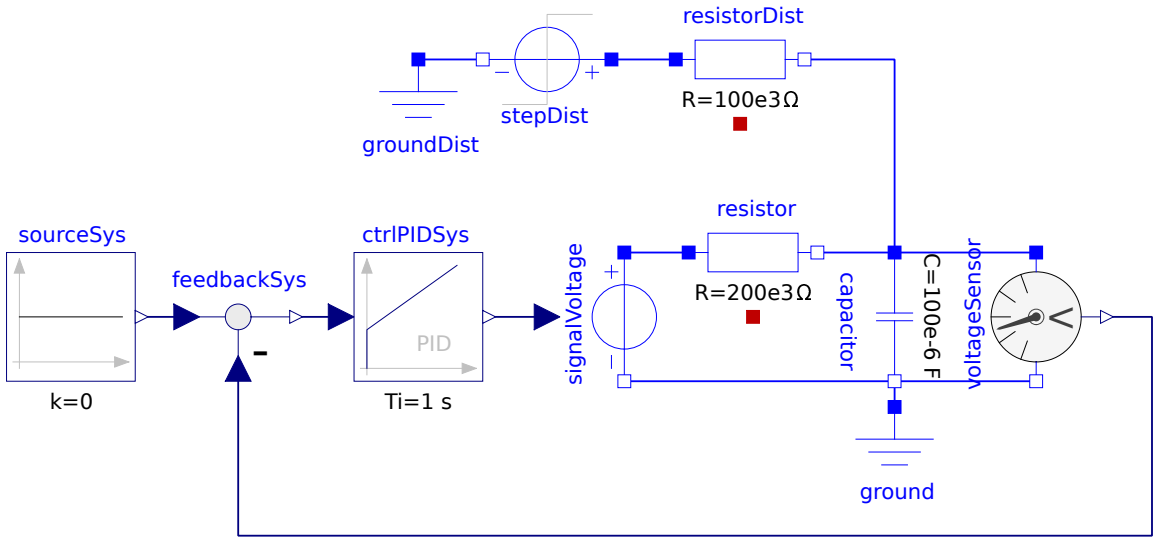


Figure 5.6: Feedback control system without feedforward part and with a physical process model.

The physical process is represented by an RC element (resistor and capacitor). The

voltage across the RC element corresponds to the manipulated variable, the voltage dropping across the capacitor to the controlled variable. An RC element represents a system with a PT1 transfer behavior. The controlled variable is disturbed by a voltage which acts on the RC element after a certain time via a resistor. Together with the capacitor, the disturbance forms another RC element with PT1 transfer behavior. The time constant τ of an RC element is calculated by:

$$\tau = R \cdot C \tag{5.1}$$

The time constant τ corresponds to the values selected in Section 2.5.1. The component dimensions are given in Figure 5.6. The controlled system and the disturbance system influence each other, which is why the system behavior with the selected dimensions is only approximately identical to that in Section 2.5.1.

The integration of the feedback control system with physical components into an *Experimental Frame* (EF) for simulation experiments is shown in Figure 5.7, which is based on Figure 3.13. The EF is essentially the same as the EF in Section 3.3.2, except that the generation of the disturbance signal is adapted. As can be seen in Figure 5.6, in addition to the process, the disturbance is also represented as a physical model. Therefore, the component of the *source disturbance* in the generator changes, while the transducer and acceptor remain identical. In Figure 5.7, a step-shaped voltage signal is selected for the source disturbance.

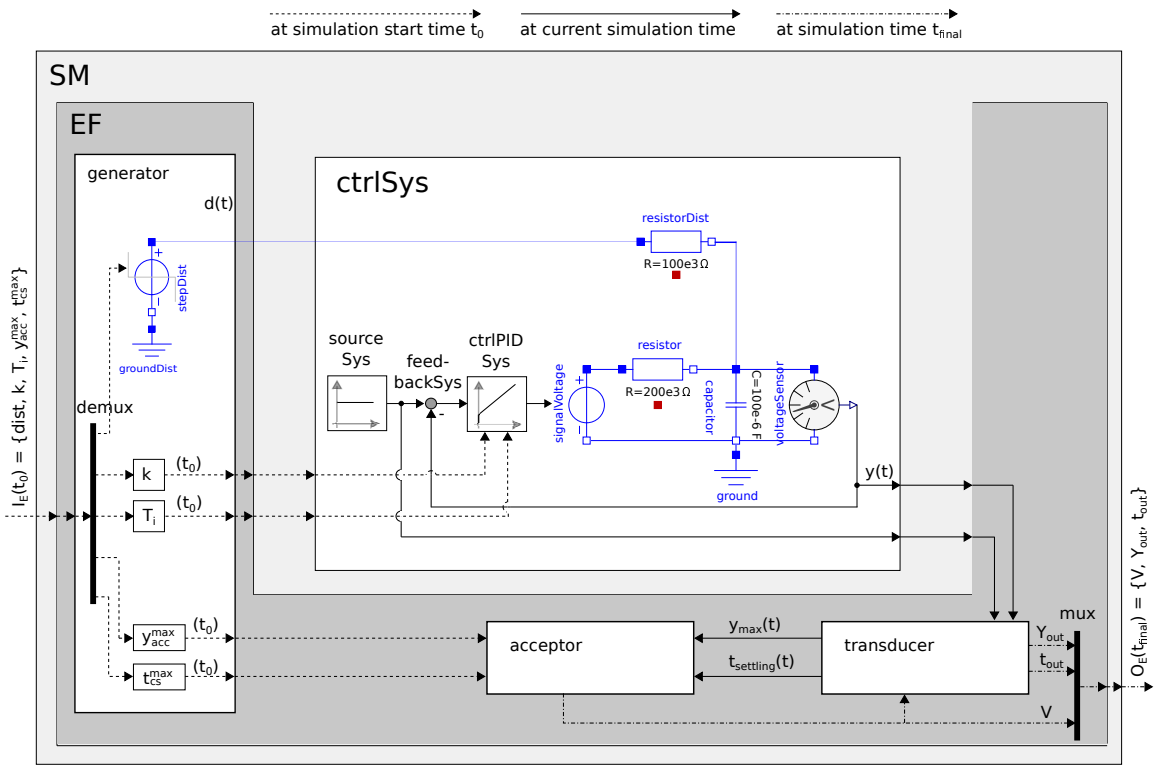


Figure 5.7: (based on Figure 3.13) Integration of the new feedback control system with physical process model in the existing EF and adaptation of the EF generator component.

In the SES/MB-based architecture for multiple simulators presented in Chapter 4, physical systems are discussed in Section 4.4.5. Thereby, the essential difference to signal flow-oriented systems in the context of model building with SESMoPy is described.

The main differences identified are that edges are non-directional and represent energy flows modeled with potential variables and flow variables. According to the Modelica specification, a third type of physical modeling variables are stream variables, which describe a bi-directional flow of matter with convective transport of specific quantities (specification in [91]). Stream variables are transmitted for example via FluidPorts. When modeling with electronic components, voltage is a potential variable and current is a flow variable. The heat dependence of resistors can be modeled using the unconnected (red) HeatPorts shown in Figures 5.6 and 5.7, which can be used to model the potential variable temperature and the heat flow. Stream variables are not needed.

The model structure shown in Figure 5.6 is based on components of the Modelica Standard Library. The use of an MB with Modelica standard components supports the generation of executable models for different Modelica target simulators. If FMI-based components – *Functional Mock-up Unit*s (FMUs) – are organized in the MB, they can be integrated into a Modelica model. To generate an overall model for different non-Modelica-based target simulators, the overall model must be exported as a *model FMU* according to the discussion in Section 4.4.5.

5.2.2 SES/MB-based Modeling of the Feedback Control System with Physical Process Model and Derivation of a Possible Flattened Pruned Entity Structure

Section 3.2.3 explained the SES/MB-based modeling of the feedback control system with a transfer function-based process and disturbance model. Figures 5.8 and 5.9 show the SES and the corresponding MB according to the model structure in Figure 5.6. The MB contains signal flow-oriented, physical, and FMI-based components. The Modelica-based components are shown with their respective icons, while the FMI-based component is shown as a file with the extension *fmu*. The integration of the *Model Under Study* (MUS) with the EF is not discussed here, as it is essentially the same as the representation in Section 3.3.2.

The former nodes *procUnitSys*, *sourceDist*, *tfDist*, and *addDist*, which describe the process and the disturbance path, are now changed in the SES to nodes for mapping the physical components. The entity node *procUnitSys* is extended by an aspect node *procUnitSysDEC*, which describes the composition of voltage source *signalVoltage*, *resistor*, *capacitor*, and *ground*. Analogously, the disturbance source *sourceDist* is composed of *stepDist* and *groundDist*. The node *tfDist* is replaced by *resistorDist* and the node *addDist* by *voltageSensor*. The nodes *expMethod* and *simMethod* correspond to the specification in Section 4.4.2. A possible FPES for the feedback control system without feedforward control part with the components of the physical submodel is shown in Figure 5.10.

At this point, the mb-attribute in the FPES is to be considered in more detail. Signal flow-oriented and physical components in an OpenModelica MB as well as FMI-based components in files are referenced. When using the FMI-based model builder, the link to an MB or an FMU basic model is given in the mb-attribute. The coupling relations of the FPES are shown in Table 5.1. Ports in the signal flow-oriented model part are called u for input ports and y for output ports. Physical ports in electrical components can be a positive pin p or a negative pin n. Exceptions are the voltage source *signalVoltage* and the

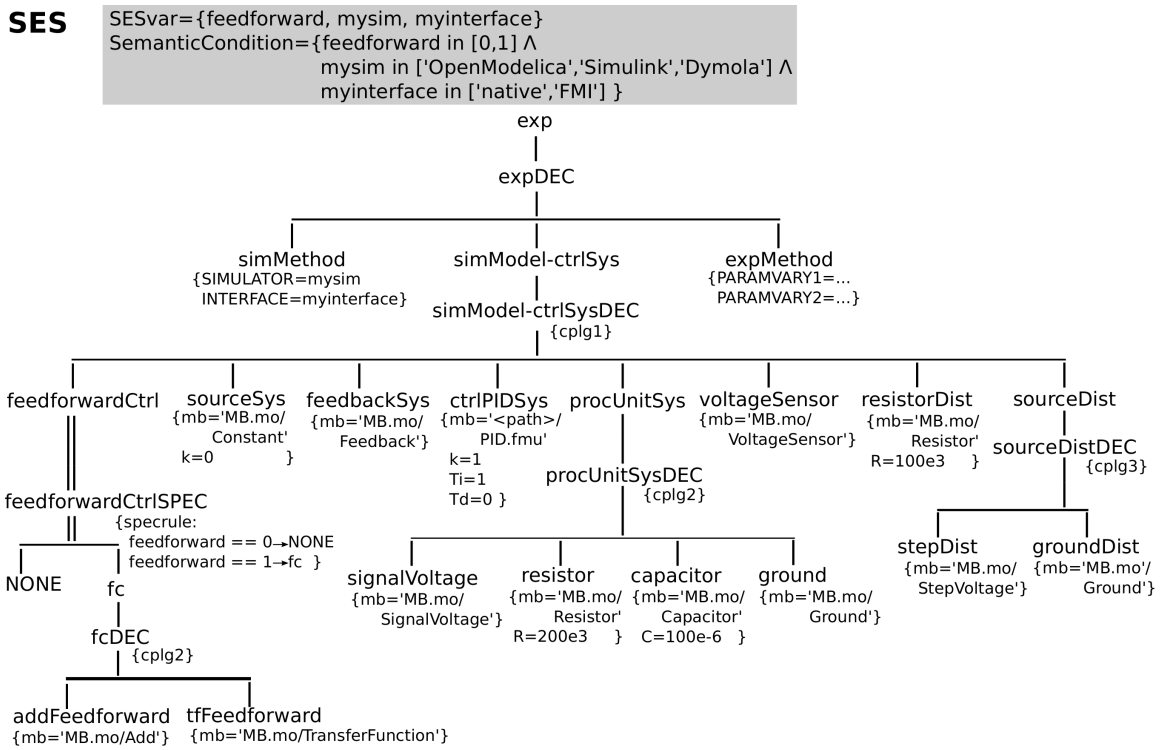


Figure 5.8: SES of the feedback control system with physical components without coupling relations.

sensor *voltageSensor*, which have both types of ports. The type of a port is specified in the signal flow-oriented model part as *Signal Port Real* (SPR) and in the physical model part as *Physical Port Electrical Analog* (PPEA). PPEA describes an electrical pin via which the potential quantity voltage and flow quantity current are transported.

Table 5.1: Coupling relations of the FPES in Figure 5.10.

Source node			*Sink node*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
ctrlPIDSys	y	SPR	signalVoltage	v	SPR
signalVoltage	p	PPEA	resistor	p	PPEA
resistor	n	PPEA	capacitor	p	PPEA
signalVoltage	n	PPEA	capacitor	n	PPEA
capacitor	n	PPEA	ground	p	PPEA
groundDist	p	PPEA	stepDist	n	PPEA
stepDist	p	PPEA	resistorDist	p	PPEA
resistorDist	n	PPEA	capacitor	p	PPEA
capacitor	p	PPEA	voltageSensor	p	PPEA
capacitor	n	PPEA	voltageSensor	n	PPEA
voltageSensor	v	SPR	feedbackSys	u2	SPR

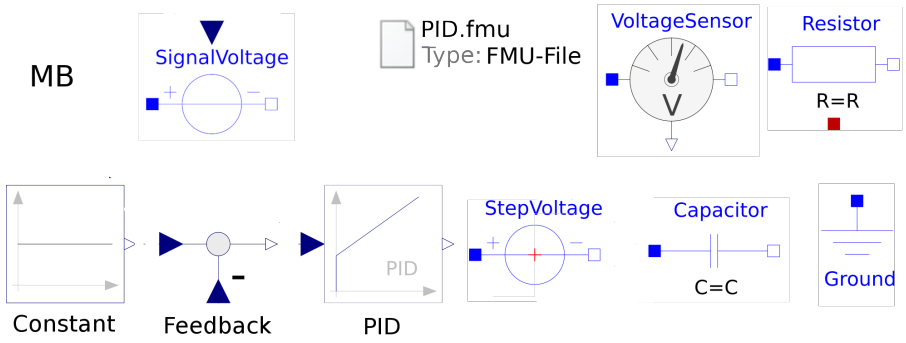


Figure 5.9: MB for the SES in Figure 5.8.

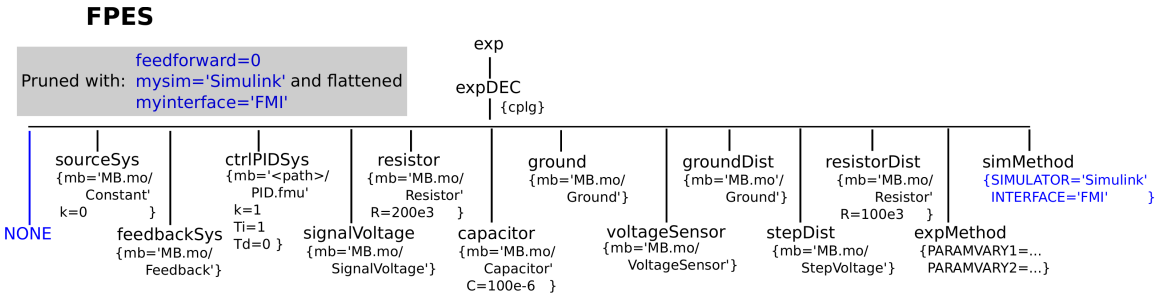


Figure 5.10: A possible FPES of the feedback control system without feedforward control with the components of the physical submodel.

5.2.3 Model Generation Using the Functional Mock-up Interface

The model builder using FMI was discussed in Section 4.4.5 using the feedback control system example as a signal flow-oriented model. FMI was motivated and developed with the focus on physical systems. The FMI-based model builder implemented in SESMoPy is used here.

An overall model is created from the FPES using the MB in OpenModelica. While Modelica-based models from an MB are configured directly in OpenModelica, FMUs are imported into OpenModelica before configuration. Then, in the overall model, input and output interfaces are added for all model quantities. The configured overall model in OpenModelica is shown in Figure 5.11.

While signal flow-oriented systems are extended by single output blocks as shown in Figure 4.9, physical systems are extended by output blocks for *all* variables transmitted over an edge. As shown in Figure 5.11, potential variables can be accessed in the connectors at a time t as a scalar value. Flow variables, on the other hand, require a physical sensor for scalar output. Since physical systems of different domains (electrical, mechanical,...) have different variables in their connectors, different sensors are needed for each physical domain. Thus, the type of the port of a coupling needs to represent the type of the physical domain. In case of an electronic system, corresponding output blocks for the potential variable *voltage* are inserted. Amperemeters for measuring the flow variable *current* are inserted in series with the electronic components and their output is provided with an output block in each case.

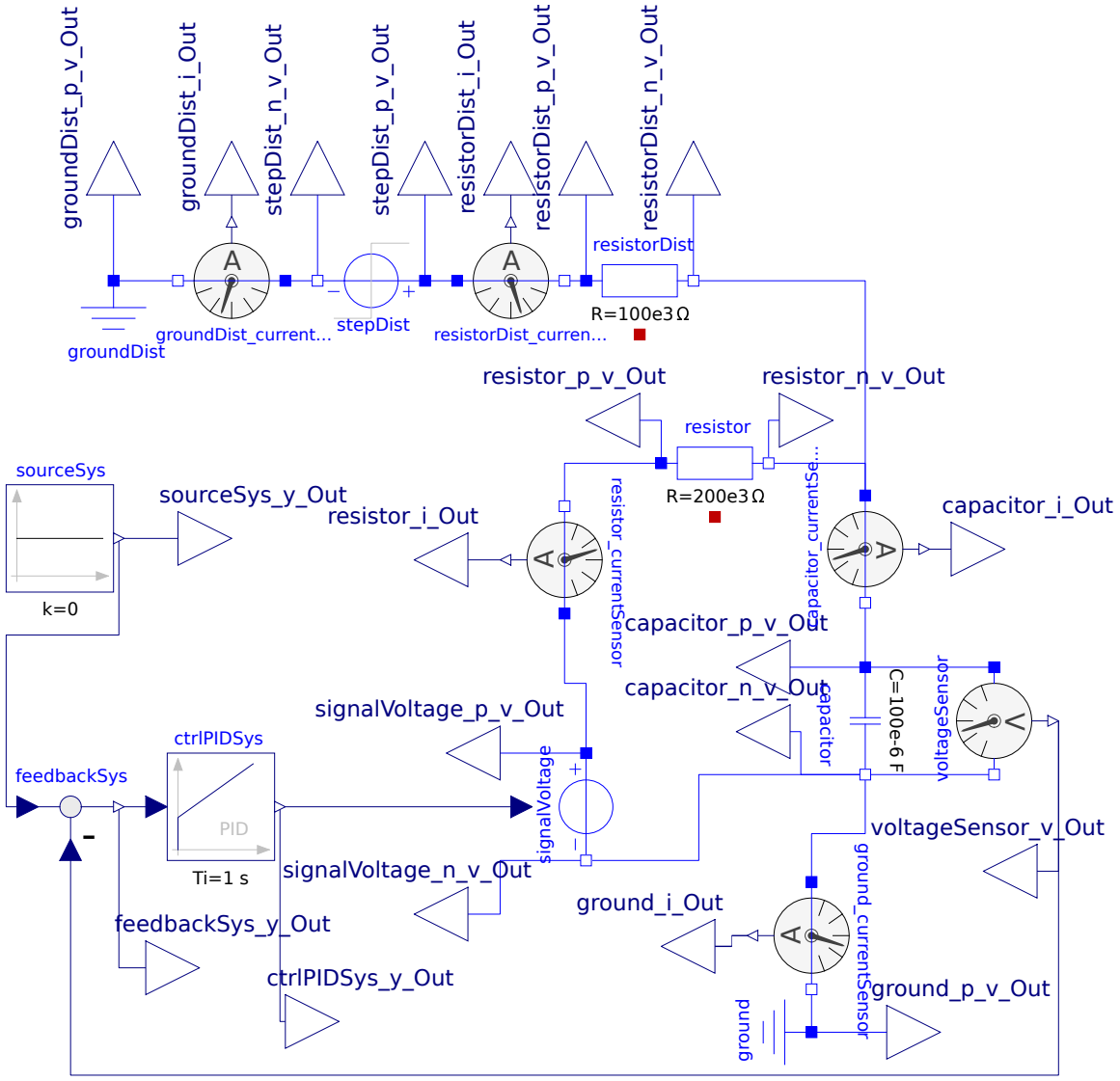


Figure 5.11: OpenModelica overall model including all configured input and output interfaces.

Subsequently, the complete model is exported as model FMU. Due to the configured interface, all model quantities calculated during the simulation and possibly required for a simulation experiment can be accessed. According to Section 4.4.5, a separate model FMU is created for each parameter variation specified in *expMethod*. Depending on the target simulator specified in *simMethod*, either *Simulation Model Executable*s (SMEs) or *Simulation Model Representation*s (SMRs) are created.

When the model is created, a configuration file is written. This file contains the information specified in the *expMethod* and *simMethod* nodes, such as the target simulator, the interface, and the parameter variations created for various model FMUs. Simulation parameters are added to the configuration file in the *Experiment Control* (EC) (Section 4.4.6) so that SMEs or SMRs can be executed in the target simulator based on the configuration file. Listing 5.1 shows the configuration file for the example model. Simulink simulation results are shown in Figure 5.12. The value of *sourceSys.y* represents the setpoint, *stepDist.p.v* the disturbance, and *voltageSensor.v* the controlled variable. The plot is similar to the top left plot in Figure 3.17.

```
1   Configuration file for model execution with SESEuPy.
2
3   SESVAR: feedforward = 1
4   SESVAR: mysim = Simulink
5   SESVAR: myinterface = FMI
6
7   MODELNAMEPARAM: Simulink_FCS_model_1 with varying parameterization: ctrlPIDSys:
        k=1 Ti=1
8   MODELNAMEPARAM: Simulink_FCS_model_2 with varying parameterization: ctrlPIDSys:
        k=5 Ti=0.5
9
10  MODEL: <path>/Simulink_FCS_model_1.m
11  MODEL: <path>/Simulink_FCS_model_2.m
12
13  SIMULATOR: Simulink
14  INTERFACE: FMI
15
16  MODELBASE: (<path>/Simulink_FCS_model_1.m) <path>/FMU_FCS_model_1.fmu
17  MODELBASE: (<path>/Simulink_FCS_model_2.m) <path>/FMU_FCS_model_2.fmu
18
19  STARTTIME: 0
20  SOLVER: ode45
21  STOPTIME: 50
22  MAXSTEP: 0.1
23  EXECTYPE: sequential
24
25  NSIGANA: ['sourceSys.y', 'stepDist.p.v', 'voltageSensor.v']
```

Listing 5.1: Configuration file for the feedback control system using the FMI approach.

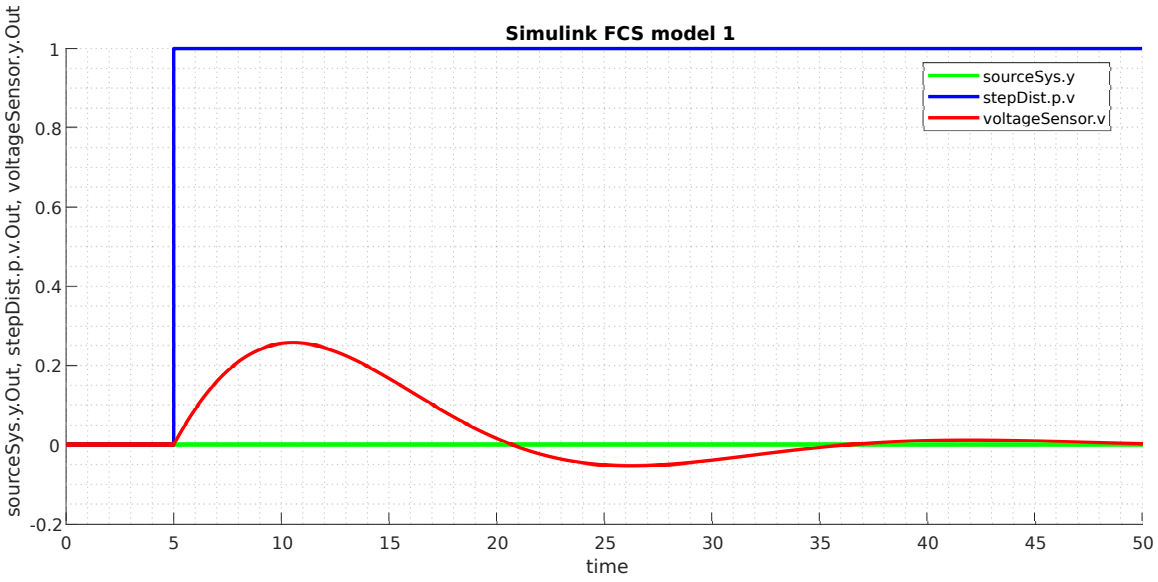


Figure 5.12: Simulation results of the overall model simulated in Simulink with the configuration file in Listing 5.1.

5.3 Summary

In this chapter, two examples of the developed SES/MB architecture for simulation with different simulators were shown.

In the first example, a concrete discrete event-oriented system with different levels of abstraction was presented. Different abstraction levels of the system were specified in an

SES. DEVS-based components were organized in an MB. DEVS-based systems represent a superclass for discrete event-oriented systems. Model generation was done according to the native approach for the DEVSimPy software.

In the second example, the feedback control system example was extended by physical components that represent a real process. Furthermore, the integration with an EF for simulation-based experiments was described. The SES for specifying the two different system structures including different types of basic systems including physical ones was discussed. To perform simulation-based experiments with the SES/MB architecture, a *Simulation Method* (SnM) and an *Experiment Method* (ExM) were specified. Signal flow-oriented, physical and FMI-based components are organized in the MB. The derivation of a possible FPES was shown. Furthermore, an OpenModelica-based overall model was generated and configured using the MB. By adding interfaces and exporting as FMU, a so-called *model FMU* was generated, which is executable in all simulators with FMI support.

With the two examples it was proven that with the introduced SES/MB architecture executable simulation models can be generated across simulators for all system classes described before. When using the FMI approach, the cross-simulator model generation can be performed on the basis of a common MB.

The methods developed for *Modeling and Simulation* (M&S) can also be used for non-simulation specific software applications. An example of this is shown in Appendix I.

6 Conclusion and Future Work

According to the formulated objectives, an approach for variant management of modular-hierarchical and multifaceted systems using different *Modeling and Simulation* (M&S) environments has been developed. Furthermore, a software architecture for M&S of *Families of Models* (FoMs) was designed, which supports automation of model selection, model generation, and simulation execution. The architecture was implemented prototypically in the form of Python-based software tools using modeling standards and model exchange standards [51]. The use of the developed methods and the software architecture was investigated on the basis of different use cases from various application areas.

The *System Entity Structure* (SES)/*Model Base* (MB) approach was selected as the basis for variant management and the derived software architecture. This has its origin in simulation theory and was introduced by Zeigler [163]. The SES describes a set of system configurations in the form of a tree structure in a *restricted simulator-independent* way, while in an MB *simulator-dependent* dynamic basic systems are organized in a reusable way as building blocks for the generation of overall models. The SES/MB approach defines general methods for goal-directed selection of system configurations (pruning method) and for generation of executable simulation models (build method). In Schmidt [135], the SES/MB approach was extended towards variant management and the generation of simulation-based experiments for a specific M&S environment. In this work, the software architecture derived by Schmidt was partially built upon.

The analysis of existing works showed the challenges to achieve the formulated objectives of this work. In the context of the SES, an extended algorithm of the pruning method for the automated derivation of system configurations was developed. Furthermore, an approach for the evaluation of hierarchical multi-aspect nodes was introduced. The extension completely removes previous limitations in the automated pruning of SES. Thus, the possibilities of SES-based modeling are significantly extended. In addition, the specification of coupling relationships in an SES has been redefined. Thus, models based on basic models of different system types from different domains, such as signal flow-oriented systems, discrete event systems, physical systems, or software systems, can be specified in an SES and generated using an MB. The investigations on the use of different M&S environments showed that in simulation-based experiments the *Experiment Method* (ExM) and *Simulation Method* (SnM) can no longer be organized in a generally valid way in an overarching MB.

In the software architecture developed in this work for the automation of simulation-based experiments using different M&S environments, the SnM and the ExM were outsourced to components of the architecture. The architecture consists of the SES/MB framework, a higher-level *Experiment Control* (EC), and an *Execution Unit* (EU). The EU provides

an interface for the execution of models in the respective target simulators. For the organization of components in an MB, two new approaches were developed as part of the architecture. In the *native* approach, simulator-specific components and *additionally* a simulator-dependent function are organized in the MB. The simulator-specific function of an MB ensures that different simulator-specific MBs have a compatible interface to connect to *one* SES. With this extension of the MBs, the specification of an SES becomes *truly* simulator-independent. The second approach uses the *Functional Mock-up Interface* (FMI) as the simulator-independent interface. In this approach, both FMI-based components and simulator-specific components of an *intermediate simulator* can be organized in an MB. An overall model is generated in the intermediate simulator, which is exported as an overall model with FMI interface. This can finally be used in any target simulator with FMI support.

The functionality of the methodological extensions and new developments as well as the developed architecture were demonstrated by means of use cases and application examples for different dynamic systems (signal flow-oriented systems, discrete event systems, and physical systems) from different domains (environmental domain, control engineering, electrical engineering). Furthermore, an example from software engineering was used to show that the developed approach is also suitable for non-simulation-specific applications.

*Cyber-Physical System*s (CPS) were mentioned in the thesis as an example of technical systems that often reach a high degree of complexity. The work itself does not examine a complete CPS in the form of a use case. However, use cases are considered that involve subsystems of CPS. In Chapter 2, a simple control system was introduced as a use case of a technical system. This contains two structure variants and considers two parameter variants as an example. In Chapter 3, SES/MB-based modeling including the concept of the *Experimental Frame* (EF) is demonstrated on this use case. Subsequently, in Chapter 4, this use case is used to demonstrate the basics of the developed software architecture. In Chapter 5, the use of the developed methods as well as the software architecture were practically examined on various use cases from different application areas and, in this context, a physical process was also added to the control system. Altogether with the considerations in the chapters two to five of the work the hypotheses one to three of the introduction were examined and stated.

Chapter 4 introduced an SES/MB-based software architecture for automating simulation experiments using various target simulators, which was implemented in a reusable manner as a software prototype. Together with the use-cases considered in Chapter 5 and follow-up uses of the software prototype at the University of Corsica, SPE Laboratory, UMR CNRS 6134 [24] and the University of Arizona, Department of Electrical & Computer Engineering (use in engineering education by Professor J. Rozenblit), hypotheses four and five of the introduction were stated.

The approach developed in this thesis supports a domain-spanning, systematic, and partly simulator-independent model development as well as model generation. There is a strict separation between the specification of dynamic behavior, model structures, parameter configurations, and model variants as well as the execution (simulation) of models. The associated software architecture supports an automation of simulation studies considering different system variants from model selection to simulation execution. The latter can be done sequentially as well as in parallel and with different target simulators. Simulation studies can be executed reactively depending on previous simulation results.

Variant studies with respect to different model structures require that the variants have been specified in advance in the SES. Parameter configurations can be re-generated during the investigation if the parameters have been modeled as variable parameters. Regarding the principle application of the introduced approach, it has to be estimated that the approach is very flexible and supports a high degree of automation. However, due to the variety of methods and software used, additional training effort is required. Accordingly, the effectiveness of the developed methods depends on the complexity and variety of a planned investigation. The approach is not suitable for limited model investigations on a small scale due to the training time.

In the thesis, the use of the developed methods was motivated in the context of the development of CPS and an application example from the environmental sector was also considered in the use cases. Typical further fields of application are seen in the planning of complex production systems and in the virtual commissioning of control systems. In both fields of application, the dynamic behavior of often complex systems is represented in models, which are used to investigate the behavior and properties of systems under various conditions. Especially when investigating control system variants, a high number of variants quickly arises. Virtual commissioning often involves testing code on virtual plant models via simulation. These tests require models with a high level of detail and incremental development of models with varying levels of detail. The watershed example in Chapter 5 demonstrates the suitability of the SES/MB-based approach to such problems.

With regard to the methodological further development of the presented approach, investigations on the cross-simulator organization of MBs are to be mentioned in particular. For continuous and physical system models, it was shown that FMI provides a solid basis for this. Furthermore, it should be investigated whether the new FMI 3.0 standard and the *System Structure & Parameterization* (SSP) standard offer advantages and simplifications with respect to the application in the context under consideration. Since there is no adequate standard for *Discrete Event System*s (DES) so far, it should be investigated to what extent the new standards are applicable to DES and alternative approaches should be researched in order to enable an organization of simulator-independent MBs for all classes of dynamic systems.

Regarding the developed software architecture and its prototypical implementation, it has to be estimated that with the implemented tool SESToPy a user-friendly and stable tool for the creation and processing of SES models is available. The Python-interpreter-based EC is also usable with Python knowledge. Further need for investigation is seen with the EU. This implements a kind of wrapper for used target simulators. The prototype implementation supports OpenModelica, Dymola, Simulink, and DEVSimPy, but does not yet support any commercial simulator for discrete event systems. OpenModelica and Dymola share the Modelica library, but the *Application Programming Interface* (API) of both programs differs. Due to the already discussed lack of a standard for cross-simulator model exchange for DES, only the integration of a *Discrete Event System Specification* (DEVS)-based simulator was investigated in this work. Analogous to a large number of colleagues in the M&S community, the author sees DEVS as a generic basis for model exchange, at least from a theoretical point of view.

Bibliography

[1] Dirk Abel and Alexander Bollig. *Rapid Control Prototyping: Methoden und Anwendungen*. Springer-Verlag Berlin Heidelberg, 01 2006. ISBN 9783540295242. doi: 10.1007/3-540-29525-9.

[2] Oliver Alt. *Modellbasierte Systementwicklung mit SysML*. Hanser, München, 2012. ISBN 9783446430662. doi: 10.3139/9783446431270.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013. ISBN 9783642375200.

[4] Art Systems. FluidSim, 2022. URL `https://www.art-systems.de/www/site/en/fluidsim`. Accessed: 2022-12-25.

[5] Osman Balci. *Verification, Validation, and Testing*, chapter 10, pages 335–393. Volume 1 of *A Wiley-Interscience publication* Banks [6], 1998. ISBN 9780471134039. doi: 10.1002/9780470172445.ch10.

[6] Jerry Banks. *Handbook of Simulation - Principles, Methodology, Advances, Applications, and Practice*. A Wiley-Interscience publication. John Wiley & Sons, Ltd, 1998. ISBN 9780471134039.

[7] Russell R. Barton. Designing simulation experiments. In *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*, WSC '13, page 342–353. IEEE Press, 2013. ISBN 9781479920778.

[8] Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for co-simulation using FMI. *8th International Modelica Conference*, 06 2011. doi: 10.3384/ecp11063115.

[9] Ola Batarseh and Leon F. McGinnis. System modeling in SysML and system analysis in Arena. In *Proceedings - Winter Simulation Conference*, pages 1–12, 12 2012. ISBN 9781467347792. doi: 10.1109/WSC.2012.6465139.

[10] Perakath C. Benjamin, Madhav Erraguntla, Dursun Delen, and Richard J. Mayer. Simulation modeling at multiple levels of abstraction. In Deborah J. Medeiros, Edward F. Watson, John S. Carson II, and Mani S. Manivannan, editors, *Proceedings of the 30th conference on Winter simulation, WSC 1998, Washington DC, USA, December 13-16, 1998*, pages 391–398. WSC, 1998. doi: 10.1109/WSC.1998.745013.

[11] Oliver Bertram. Model-based design of safety-critical aircraft systems. ARGESIM Report 45, ARGESIM Pub. Vienna, Austria, 2021.

[12] María Julia Blas, Silvio Gonnet, and Bernard P. Zeigler. Towards a universal representation of DEVS: A metamodel-based definition of DEVS formal specification. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*, pages 1–12, 2021. doi: 10.23919/ANNSIM52504.2021.9552162.

[13] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, Dietmar Neumerkel, Hans Olsson, Jörg-Volker Peetz, Susann Wolf, and Christoph Clauß. The Functional Mockup Interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference*, pages 105–114, 2011. doi: 10.3384/ecp11063105.

[14] Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International Modelica Conference*, pages 173–184. The Modelica Association, 2012. ISBN 9789175198262. doi: 10.3384/ecp12076173.

[15] Jean-Sébastien Bolduc and Hans Vangheluwe. The modelling and simulation package PythonDEVS for classical hierarchical DEVS. *MSDL Technical Report MSDL-TR-2001–01, McGill University*, 2001.

[16] Spencer Borland and Hans Vangheluwe. Transforming statechart models to DEVS. Technical report, School of Computer Science, McGill University, Montréal, Canada, 2003.

[17] Hartmut Bossel. *Simulation dynamischer Systeme*. Vieweg+Teubner, 1st edition, 1989. ISBN 9783528047467.

[18] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*, volume 1. Morgan & Claypool Publishers, 09 2012. doi: 10.2200/S00441ED1V01Y201208SWE001.

[19] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice: Second Edition*. Morgan & Claypool Publishers, 2nd edition, 2017. ISBN 9781627057080.

[20] Felix Breitenecker. Models, methods and experiments - a new structure for simulation systems. *Mathematics and Computers in Simulation*, 34(3):231–260, September 1992. ISSN 0378-4754. doi: 10.1016/0378-4754(92)90004-z.

[21] Lena Buffoni, Lennart Ochel, Adrian Pop, Peter Fritzson, Niklas Fors, Görel Hedin, Walid Taha, and Martin Sjölund. Open source languages and methods for cyber-physical system development: Overview and case studies. *Electronics*, 10(8), 2021. ISSN 2079-9292. doi: 10.3390/electronics10080902.

[22] Laurent Capocchi. DEVSimPy. `https://github.com/capocchi/DEVSimPy`, 2019.

[23] Laurent Capocchi, Jean-François Santucci, Bastien Poggi, and Céline Nicolai. DEVSimPy: A collaborative Python software for modeling and simulation of

DEVS systems. In *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 170–175, 2011. doi: 10.1109/WETICE.2011.31.

[24] Laurent Capocchi, Jean François Santucci, Thorsten Pawletta, Hendrik Folkerts, and Bernard P. Zeigler. Discrete-event simulation model generation based on activity metrics. *Simulation Modelling Practice and Theory*, 103:102122, 2020. doi: 10.1016/j.simpat.2020.102122.

[25] Yolanda Carson and Anu Maria. Simulation optimization: Methods and applications. In *Winter Simulation Conference Proceedings*, volume 1, pages 118–126, 1997. doi: 10.1109/WSC.1997.640387.

[26] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer, New York, 2nd edition, 2008. ISBN 9780387333328. doi: 10.1007/978-0-387-68612-7.

[27] CATIA-Systems. FMPy, 2021. URL `https://github.com/CATIA-Systems/FMPy`. Accessed: 2021-10-15.

[28] François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer US, New York, NY, 1st edition, 2006. ISBN 9781441938633. doi: 10.1007/0-387-30260-3.

[29] Deniz Cetinkaya, Alexander Verbraeck, and Mamadou Seck. MDD4MS: A model driven development framework for modeling and simulation. In *Proceedings of the 2011 Summer Computer Simulation Conference*, pages 113–121, 06 2011.

[30] Deniz Cetinkaya, Alexander Verbraeck, and Mamadou D. Seck. Model transformation from BPMN to DEVS in the MDD4MS framework. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, TMS/DEVS '12, San Diego, CA, USA, 2012. Society for Computer Simulation International. ISBN 9781618397867.

[31] Wuzhu Chen, Michaela Huhn, and Peter Fritzson. A generic FMU interface for Modelica. In François E. Cellier, David Broman, Peter Fritzson, and Edward A. Lee, editors, *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2011, Zurich, Switzerland, September 5, 2011*, volume 56 of *Linköping Electronic Conference Proceedings*, pages 19–24. Linköping University Electronic Press, 2011.

[32] J. Chew and C. Sullivan. Verification, validation, and accreditation in the life cycle of models and simulations. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 1, pages 813–818 vol.1, 2000. doi: 10.1109/WSC.2000.899879.

[33] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. Abstract delta modeling. *SIGPLAN Not.*, 46(2):13–22, October 2010. ISSN 0362-1340. doi: 10.1145/1942788.1868298.

[34] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[35] The AnyLogic Company. Anylogic, 2021. URL `https://www.anylogic.com`. Accessed: 2021-10-15.

[36] L. Coron, V. Andréassian, C. Perrin, M. Bourqui, and F. Hendrickx. On the lack of robustness of hydrologic models regarding water balance simulation: a diagnostic approach applied to three models of increasing complexity on 20 mountainous catchments. *Hydrology and Earth System Sciences*, 18(2):727–746, 2014. doi: 10.5194/hess-18-727-2014.

[37] CPS Steering Group. Cyber-physical systems - executive summary, 2011. URL `http://iccps.acm.org/2011/_doc/CPS-Executive-Summary.pdf`. Accessed: 2021-10-15.

[38] Dassault Systèmes. Dymola, 2021. URL `https://www.3ds.com/products-services/catia/products/dymola/`. Accessed: 2021-10-15.

[39] Thorsten Daum and Robert G. Sargent. Experimental frames in a modern modeling and simulation system. *IIE Transactions*, 33(3):181–192, 2001. doi: 10.1080/07408170108936821.

[40] Christina Deatcu, Hendrik Folkerts, Thorsten Pawletta, and Umut Durak. How to define SES trees for variability modeling. *SNE Simulation Notes Europe*, 29(3):117–126, 2019. doi: 10.11128/SNE.29.TN.10482.

[41] Joachim Denil, Stefan Klikovits, Pieter J. Mosterman, Antonio Vallecillo, and Hans Vangheluwe. The experiment model and validity frame in M&S. In Fernando Barros, Xiaolin Hu, Joachim Denil, and Rhys Goldstein, editors, *Proceedings of the Symposium on Theory of Modeling & Simulation*, TMS/DEVS '17, pages 10:1–10:12, San Diego, CA, USA, April 2017. Society for Computer Simulation International.

[42] Der Beauftragte der Bundesregierung für Informationstechnik. Das V-Modell XT, 2005. URL `https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html`. Accessed: 2021-10-15.

[43] Arie Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10, 01 2002. doi: 10.2498/cit.2002.01.01.

[44] Umut Durak. *Model-based simulation systems engineering.* PhD thesis, Clausthal University of Technology, Aug 2018. URL `https://doi.org/10.21268/20180827-101019`.

[45] Umut Durak, Halit Oğuztüzün, and S. Kemal Ider. An ontology for trajectory simulation. In *Proceedings of the 2006 Winter Simulation Conference*, pages 1160–1167, 2006. doi: 10.1109/WSC.2006.323207.

[46] Umut Durak, Thorsten Pawletta, Halit Oğuztüzün, and Bernard P. Zeigler. System entity structure and model base framework in model based engineering of simulations for technical systems. In *SpringSim*, 04 2017. doi: 10.22360/springsim.2017.mod4sim.001.

[47] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Object-oriented and hybrid modeling in Modelica. *Journal Européen des systèmes automatisés*, 35(4): 395–404, Juli 2001. LIDO-Berichtsjahr=2000.

[48] ESI. SimulationX, 2022. URL `https://www.esi-group.com/products/system-simulation`. Accessed: 2022-12-25.

[49] Paul A. Fishwick. *Simulation model design and execution - building digital worlds*. Prentice Hall, 1995. ISBN 9780130986092.

[50] Paul A. Fishwick and Kangsun Lee. Two methods for exploiting abstraction in systems. In *AI, Simulation and Planning in High Autonomous Systems*, pages 257–264, 1996.

[51] Hendrik Folkerts. SES/MB architecture supporting several simulation tools. `https://github.com/hendrikfolkerts`, 2021. Accessed: 2021-10-15.

[52] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, Jean-François Santucci, and Laurent Capocchi. An integrated modeling, simulation and experimentation environment in Python based on SES/MB and DEVS. In *Proceedings of the 2019 Summer Simulation Conference*, SummerSim '19, San Diego, CA, USA, 2019. Society for Computer Simulation International.

[53] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, and Bernard P. Zeigler. Automated, reactive pruning of System Entity Structures for simulation engineering. In *Proceedings of the 2020 Spring Simulation Conference*, SpringSim '20, San Diego, CA, USA, 2020. Society for Computer Simulation International. ISBN 9781713812883.

[54] Rüdiger Franke, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. Discrete-time models for control applications with FMI. In *12th International Modelica Conference*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2017. doi: 10.3384/ecp17132507.

[55] Richard Fujimoto. Parallel and distributed simulation. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, page 45–59. IEEE Press, 2015. ISBN 9781467397414.

[56] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley-VCH, January 2000. ISBN 978-0-471-18383-9.

[57] Marcus Geimer, Thomas Krüger, and Peter Linsel. Co-simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung. *O+P Ölhydraulik und Pneumatik*, 50(11-12):572 – 576, 2006. ISSN 0341-2660.

[58] Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro Gonzalez-Calero, Paul Groth, Joshua Moody, and Ewa Deelman. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, 26(1):62–72, 2010. ISSN 1541-1672. doi: 10.1109/MIS.2010.9.

[59] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: A survey. *ACM Comput. Surv.*, 51(3), May 2018. ISSN 0360-0300. doi: 10.1145/3179993.

[60] Geoffrey Gordon. *The Development of the General Purpose Simulation System (GPSS)*, page 403–426. Association for Computing Machinery, New York, NY, USA, 1978. ISBN 0127450408. doi: 10.1145/800025.1198386.

[61] Ian Gorton. *Essential Software Architecture (2. ed.)*. Springer-Verlag Berlin Heidelberg, 01 2011. ISBN 9783642191756. doi: 10.1007/978-3-642-19176-3.

[62] Hans Grönniger, Holger Krahn, Claas Pinkernell, and Bernhard Rumpe. Modeling variants of automotive systems using views. In *Tagungsband Modellierungs-Workshop MBEFF: Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen.*, 03 2008.

[63] B. Gutekunst and J. Weiland. Handhabung von Varianz in Simulink aus funktionsorientierter Sicht. In *GI-Jahrestagung*, 2011.

[64] Olaf Hagendorf. *Simulation Based Parameter and Structure Optimisation of Discrete Event Systems*. PhD thesis, Liverpool John Moores University, 2009.

[65] Olaf Hagendorf and Thorsten Pawletta. *A Framework for Simulation-Based Structure and Parameter Optimization of Discrete-Event Systems*, pages 199 – 222. CRC Press Inc. of Tailor & Francis Group, 01 2011. ISBN 9781420072334. doi: 10.1201/b10412-11.

[66] Olaf Hagendorf, Thorsten Pawletta, and Roland Larek. An approach to simulation-based parameter and structure optimization of MATLAB/Simulink models using evolutionary algorithms. *SIMULATION*, 89(9):1115–1127, 2013. doi: 10.1177/0037549713500066.

[67] Jun Han, John A. Miller, and Gregory A Silver. SoPT: Ontology for simulation optimization for scientific experiments. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2909–2920, 2011. doi: 10.1109/WSC.2011.6147994.

[68] Bernhard Heinzl. *Methods for Hybrid Modeling and Simulation-Based Optimization in Energy-Aware Production Planning*. PhD thesis, TU Vienna, 01 2020.

[69] Hexagon AB. Adams, 2022. URL `https://hexagon.com/en/products/product-groups/computer-aided-engineering-software/adams`. Accessed: 2022-12-25.

[70] Shafagh Jafer, Umut Durak, Hakan Aydemir, Richard Ruff, and Thorsten Pawletta. *Advances in Software Engineering and Aeronautics*, pages 87–102. Springer International Publishing, 2018. doi: 10.1007/978-3-319-75058-3_7.

[71] Andreas Junghanns and Torsten Blochwitz. FMI is great, but not magic, 2017. URL `https://fmi-standard.org/literature`. FMI User Meeting, 12th International Modelica Conference, May 15–17, 2017, Prague, Czech Republic, Accessed: 2021-10-15.

[72] Andreas Junghanns, Torsten Blochwitz, Christian Bertsch, Torsten Sommer, Karl Wernersson, Andreas Pillekeit, Irina Zacharias, Matthias Blesken, Pierre R. Mai, Klaus Schuch, Christian Schulze, Cláudio Gomes, and Masoud Najafi. The Functional Mock-up Interface 3.0 - new features enabling new applications. In *Proceedings of the 2021 Modelica Conference*, 2021.

[73] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[74] George-Dimitrios Kapos, Vassilis Dalakas, Mara Nikolaidou, and Dimosthenis Anagnostopoulos. An integrated framework for automated simulation of SysML models using DEVS. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 90(6):717–744, 2014. doi: 10.1177/0037549714533842.

[75] Jack P. C. Kleijnen. *Experimental Design for Sensitivity Analysis, Optimization, and Validation of Simulation Models*, chapter 6, pages 173–223. Volume 1 of *A Wiley-Interscience publication* Banks [6], 1998. ISBN 9780470172445. doi: 10.1002/9780470172445.ch6.

[76] Jack P. C. Kleijnen. *Design and Analysis of Simulation Experiments*. Springer International Publishing, 2nd edition, 2008. ISBN 9783319180861. doi: 10.1007/978-3-319-18087-8.

[77] Ernesto Kofman and Sergio Junco. Quantized-state systems: A DEVS approach for continuous system simulation. *Transactions of the Society for Computer Simulation International*, 18:123–132, 09 2001.

[78] Jochen Köhler. MA-project "System Structure and Parameterization" – early insights, 2016. URL `https://ssp-standard.org/publications/2016_ModelicaConference_Tokyo/2016_Japanese-Modelica-Conference_SSP-Early-Insights.pdf`. Accessed: 2020-11-03.

[79] Carsten Kolassa, Holger Rendel, and Bernhard Rumpe. Evaluation of variability concepts for Simulink in the automotive domain. *2015 48th Hawaii International Conference on System Sciences*, pages 5373–5382, 2015.

[80] Brock J. LaMeres. *Introduction to Logic Circuits & Logic Design with VHDL*. Springer Cham, 2nd edition, 03 2019. ISBN 9783030124892.

[81] Averill M. Law and David M. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd edition, 1999. ISBN 0070592926.

[82] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. The MIT Press, 2nd edition, 2016. ISBN 9780262533812.

[83] Stefan Leye. *Toward Guiding Simulation Experiments*. PhD thesis, University Rostock, 01 2014.

[84] Samuel J. Mason. Feedback theory - some properties of signal flow graphs. *Proceedings of the IRE*, 41(9):1144–1156, 1953. doi: 10.1109/JRPROC.1953.274449.

[85] Major Masys, Martijn van Emmerik, and Pierre Bouc. Verification, validation and accreditation (VV&A)-leveraging international initiatives. *Meeting Proceedings RTO-MP-MSG-045*, 01 2006.

[86] MathWorks. Variant Manager Overview, 2021. URL `https://de.mathworks.com/help/simulink/gui/variant-manager-interface.html`. Accessed: 2021-10-15.

[87] MathWorks. Simulink, 2021. URL `https://de.mathworks.com/products/simulink.html`. Accessed: 2021-10-15.

[88] MathWorks. Simulink Aerospace Blockset, 2022. URL `https://de.mathworks.com/help/aeroblks/index.html`. Accessed: 2022-12-25.

[89] Stephen J. Mellor and Marc J. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley, 05 2002. ISBN 9780201748048.

[90] Saurabh Mittal and José Luis Risco-Martín. Model-driven systems engineering for netcentric system of systems with DEVS unified process. *2013 Winter Simulations Conference (WSC)*, pages 1140–1151, 2013.

[91] Modelica Association. Modelica, 2021. URL `https://www.modelica.org`. Accessed: 2021-10-15.

[92] Modelica Association Project FMI. Functional Mock-up Interface, 2021. URL `https://fmi-standard.org`. Accessed: 2021-10-15.

[93] Modelica Association Project FMI. Tools supporting FMI, 2021. URL `https://fmi-standard.org/tools/`. Accessed: 2021-10-15.

[94] Modelon AB. FMI library, 2021. URL `https://github.com/modelon-community/fmi-library`. Accessed: 2021-10-15.

[95] Modelon AB. FMU compliance checker, 2021. URL `https://github.com/modelica-tools/FMUComplianceChecker`. Accessed: 2021-10-15.

[96] J. Müller, Klaas Gadeyne, Mike Nicolai, and Herman Van der Auweraer. Automatic generation of simulation models for early stage evaluation of physical system topologies (WIP). *Simulation Series*, 47:221–226, 01 2015.

[97] Wolfgang Müller and Edmund Widl. Linking FMI-based components with discrete event systems. In *2013 IEEE International Systems Conference (SysCon)*, pages 676–680, 2013. doi: 10.1109/SysCon.2013.6549955.

[98] Wolfgang Müller and Edmund Widl. Using FMI components in discrete event systems. *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6, 2015. doi: 10.1109/MSCPES.2015.7115397.

[99] Object Management Group. OMG Unified Modeling Language (OMG UML), 2017. URL `https://www.omg.org/spec/UML/2.5.1/PDF`. Accessed: 2021-10-15.

[100] Object Management Group. OMG Meta Object Facility (MOF) core specification, 2019. URL `https://www.omg.org/spec/MOF/2.5.1/PDF`. Accessed: 2021-10-15.

[101] Object Management Group. MDA specifications, 2021. URL `https://www.omg.org/mda/specs.htm`. Accessed: 2021-10-15.

[102] Object Management Group. Semantics of a foundational subset for executable UML models (fUML), 2021. URL `https://www.omg.org/spec/FUML/1.5/PDF`. Accessed: 2021-10-15.

[103] Open Source Modelica Consortium. OpenModelica, 2021. URL `https://openmodelica.org`. Accessed: 2021-10-15.

[104] Opnet projects. Opnet, 2022. URL `https://opnetprojects.com/opnet-network-simulator`. Accessed: 2022-12-25.

[105] Sebastian Oster. *Feature model-based software product line testing*. PhD thesis, Darmstadt University of Technology, Germany, 2011.

[106] M. Otter. DSblock: A neutral description of dynamic systems. Version 3.1 and version 3.2. Technical report, DLR, 1992. LIDO-Berichtsjahr=1992.

[107] Martin Otter and Hilding Elmqvist. The DSblock model interface for exchanging model components. In *EUROSIM '95 Simulation Congress, 11.-15. Sept. 1995, Wien*, pages 505–510. F. Breitenecker, I. Husinsky, TU Wien, Elsevier 1995, Amsterdam, 1995. LIDO-Berichtsjahr=1995.

[108] Thorsten Pawletta and Hendrik Folkerts. Model behavior generation for multiple simulators. In Tuncer I. Ören, Bernhard P. Zeigler, and Andreas Tolk, editors, *SCS MSBoK Guide – Body of Knowledge of Modeling and Simulation Guide*, volume 1, page 4. Society for Computer Simulation Publications, San Diego/Ca, USA, 2023. accepted publication 11/2020.

[109] Thorsten Pawletta, Daniel Pascheka, Artur Schmidt, and Sven Pawletta. Ontology-assisted system modeling and simulation within MATLAB/Simulink. *SNE Simulation Notes Europe*, 24:59–68, 08 2014. doi: 10.11128/sne.24.tn.10241.

[110] Thorsten Pawletta, Artur Schmidt, Umut Durak, and Bernard P. Zeigler. A framework for the metamodeling of multi-variant systems and reactive simulation model generation and execution. In *23. Symposium Simulationstechnik*, Dresden, Germany, September 2016.

[111] Thorsten Pawletta, Artur Schmidt, Bernard P. Zeigler, and Umut Durak. Extended variability modeling using System Entity Structure ontology within MATLAB/Simulink. In *Proceedings of the 49th Annual Simulation Symposium*, ANSS '16, pages 62–69, San Diego, CA, USA, 2016. Society for Computer Simulation International. ISBN 9781510823167.

[112] Thorsten Pawletta, Artur Schmidt, and Peter Junglas. A multimodeling approach for the simulation of energy consumption in manufacturing. *Simulation Notes Europe*, 27(2):115–124, 2017. doi: 10.11128/sne.27.tn.10377.

[113] Thorsten Pawletta, Artur Schmidt, Umut Durak, and Berhard Zeigler. A framework for the metamodeling of multivariant systems and reactive simulation model generation and execution. *SNE Simulation Notes Europe*, 28(1):11–18, 03 2018. doi: 10.11128/sne.28.tn.10402.

[114] Thorsten Pawletta, Umut Durak, and Artur Schmidt. Modeling and simulation of versatile technical systems using an extended System Entity Structure/Model Base infrastructure. In Lin Zhang, Bernard P. Zeigler, and LaiLi Yuanjun, editors, *Model Engineering for Simulation*, pages 393–418. Academic Press, March 2019. doi: 10.1016/B978-0-12-813543-3.00018-4.

[115] Mikel D. Petty. *Verification, Validation, and Accreditation*, chapter 10, pages 325–372. John Wiley & Sons, Ltd, 03 2010. ISBN 9780470486740. doi: 10.1002/9780470590621.ch10.

[116] L R Petzold. Description of DASSL: a differential/algebraic system solver. *IMACS World Congress*, 9 1982.

[117] Régis Plateaux, Jean-Yves Choley, Olivia Penas, and Alain Rivière. Towards an integrated mechatronic design process. *2009 IEEE International Conference on Mechatronics*, pages 1–6, 2009. doi: 10.1109/ICMECH.2009.4957237.

[118] Herbert Prähofer. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation – Dissertationen der Johannes Kepler-Universität Linz*. Wien : VWGÖ, 1992. ISBN 9783853698747.

[119] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[120] Pure Systems. pure::variants, 2021. URL `https://www.pure-systems.com`. Accessed: 2021-10-15.

[121] QTronic. FMU SDK, 2021. URL `https://github.com/qtronic/fmusdk`. Accessed: 2021-10-15.

[122] Markus Rabe, Sven Spieckermann, and Sigrid Wenzel. *Verifikation und Validierung für die Simulation in Produktion und Logistik - Vorgehensmodelle und Techniken*. Springer Berlin Heidelberg, 2008. ISBN 9783540352815. doi: 10.1007/978-3-540-35282-2.

[123] Axel Reichwein, Christiaan J. J. Paredis, Arquimedes Canedo, Petra Witschel, Philipp Emanuel Stelzig, Anjelika Votintseva, and Rainer Wasgint. Maintaining consistency between system architecture and dynamic system models with SysML4Modelica. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM '12, page 43–48, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450318051. doi: 10.1145/2508443.2508451.

[124] José Risco-Martín, Saurabh Mittal, Bernard Zeigler, and Jesús de la Cruz. From UML state charts to DEVS state machines using XML. In *Proceedings of IEEE/ACM International Conference on Model-Driven Engineering Languages and Systems, Multi-paradigm Modeling*. Elsevier Science B. V., 01 2007.

[125] Rockwell Automation. Arena simulation software, 2021. URL `https://www.rockwellautomation.com/de-de/products/software/arena-simulation.html`. Accessed: 2021-10-15.

[126] Jerzy Rozenblit and Yueh-Min Huang. Rule-based generation of model structures in multifaceted modeling and system design. *INFORMS Journal on Computing*, 3: 330–344, 11 1991. doi: 10.1287/ijoc.3.4.330.

[127] Jerzy W. Rozenblit. Experimental frame specification methodology for hierarchical simulation modeling. *International Journal of General Systems*, 19(3):317–336, 1991. doi: 10.1080/03081079108935180.

[128] Jerzy W. Rozenblit and Bernard P. Zeigler. Representing and constructing system specifications using the System Entity Structure concepts. In *Proceedings of the 25th Conference on Winter Simulation*, WSC '93, page 604–611, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 078031381X. doi: 10.1145/256563.256742.

[129] Andreas Ruscheinski, Tom Warnke, and Adelinde M. Uhrmacher. Artifact-based workflows for supporting simulation studies. *IEEE Transactions on Knowledge and Data Engineering*, 32(6):1064–1078, 2020. doi: 10.1109/TKDE.2019.2899840.

[130] Teodora Sanislav and Liviu Miclea. Cyber-physical systems - concept, challenges and research areas. *Control Engineering and Applied Informatics*, 14:28–33, 05 2012.

[131] Jean-François Santucci, Laurent Capocchi, and Bernard P. Zeigler. System Entity Structure extension to integrate abstraction hierarchies and time granularity into DEVS modeling and simulation. *Simulation*, 92(8):747–769, August 2016. ISSN 0037-5497. doi: 10.1177/0037549716657168.

[132] Robert G. Sargent. Verification and validation of simulation models. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 183–198, 01 2011. doi: 10.1109/WSC.2011.6147750.

[133] Ina Schaefer. Variability modelling for model-driven development of software product lines. In David Benavides, Don S. Batory, and Paul Grünbacher, editors, *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, volume 37 of *ICB-Research Report*, pages 85–92. Universität Duisburg-Essen, 2010.

[134] Ina Schaefer, Alexander Worret, and Arnd Poetzsch-Heffter. A model-based framework for automated product derivation. In Goetz Botterweck, Iris Groher, Andreas Polzer, Christa Schwanninger, Steffen Thiel, and Markus Voelter, editors, *Proceedings of the 1st International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009), collocated with the 13th International Software Product Line Conference (SPLC 2009), San Francisco, USA, August 24, 2009*, volume 557 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[135] Artur Schmidt. *Variantenmanagement in der Modellbildung und Simulation unter Verwendung des SES/MB Frameworks*. PhD thesis, University Rostock, 06 2019.

[136] Artur Schmidt and Thorsten Pawletta. Ein Ontologie-basierter Modellierungs- und Simulationsansatz am Beispiel der ressourceneffizienten Planung spanender Prozessketten. In *ASIM SPL 2013*, Paderborn, Germany, 2013.

[137] Artur Schmidt, Umut Durak, and Thorsten Pawletta. Model-based testing methodology using System Entity Structures for MATLAB/Simulink models. *SIMULATION*, 92(8):729–746, 2016. doi: 10.1177/0037549716656791.

[138] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, February 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.58.

[139] Tobias Schwatinski and Thorsten Pawletta. Ontologische Modellierung und Modellgenerierung in der MATLAB/Simulink Umgebung: Die Tiny SES Toolbox. In *ASIM STS/GMMS 2013*, Düsseldorf, Germany, 2013.

[140] Jon C. Strauss, Donald C. Augustin, Mark S. Fineberg, Bruce B. Johnson, Robert N. Linebarger, and F. John Sansom. The SCi continuous system simulation language (CSSL). *SIMULATION*, 9(6):281–303, 1967. doi: 10.1177/003754976700900601.

[141] Technische Hochschule Mittelhessen. Parametric diagram, 2021. URL `https://wiki.thm.de/Parametric_Diagram`. Accessed: 2021-10-15.

[142] Yentl Van Tendeloo and Hans Vangheluwe. PythonPDEVS: a distributed parallel DEVS simulator. In Fernando Barros, Moon Ho Wang, Herbert Prähofer, and Xiaolin Hu, editors, *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, part of the 2015 Spring Simulation Multiconference, SpringSim '15, Alexandria, VA, USA, April 12-15, 2015*, pages 91–98. SCS/ACM, 04 2015.

[143] Alejandro Teran-Somohano, Alice E. Smith, Joseph Ledet, Levent Yilmaz, and Halit Oğuztüzün. A model-driven engineering approach to simulation experiment design and execution. In *2015 Winter Simulation Conference (WSC)*, pages 2632–2643, 2015. doi: 10.1109/WSC.2015.7408371.

[144] Michael Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, Boston, 2001. ISBN 0792373677.

[145] Okan Topçu, Umut Durak, Halit Oğuztüzün, and Levent Yilmaz. *Distributed Simulation - A Model Driven Engineering Approach*. Springer International Publishing, 1st edition, 02 2016. ISBN 9783319030494. doi: 10.1007/978-3-319-03050-0.

[146] Mamadou Traoré and Alexandre Muzy. Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory*, 14: 126–142, 02 2006. doi: 10.1016/j.simpat.2005.03.002.

[147] Anargyros Tsadimas, George-Dimitrios Kapos, Vassilis Dalakas, Mara Nikolaidou, and Dimosthenis Anagnostopoulos. Simulating simulation-agnostic SysML models for enterprise information systems via DEVS. *Simulation Modelling Practice and Theory*, 66:243–259, 2016. ISSN 1569-190X. doi: 10.1016/j.simpat.2016.04.001.

[148] Adelinde M. Uhrmacher. Seven pitfalls in modeling and simulation research. In *Proceedings of the Winter Simulation Conference*, WSC '12. Winter Simulation Conference, 2012.

[149] Adelinde M. Uhrmacher and Danny Weyns. Multi-agent systems - simulation and applications. In *Multi-Agent Systems*, page 566, Boca Raton, 2009. CRC Press. ISBN 9781315218786. doi: 10.1201/9781420070248.

[150] University of Applied Sciences Wismar, Research Group CEA. hyPDEVS toolbox for MATLAB (MatlabDEVS Tbx), 2021. URL `https://www.cea-wismar.de/tbx/DEVS_Tbx/MatlabDEVS_Tbx.html`. Accessed: 2021-10-15.

[151] Herman Van der Auweraer, Jan Anthonis, Stijn Bruyne, and Jan Leuridan. Virtual engineering at work: The challenges for designing mechatronic products. *Engineering with Computers*, 29, 07 2012. doi: 10.1007/s00366-012-0286-6.

[152] Hans Vangheluwe. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *CACSD. Conference Proceedings. IEEE International Symposium on Computer-Aided Control System Design (Cat. No.00TH8537)*, pages 129–134, 2000. doi: 10.1109/CACSD.2000.900199.

[153] Hans Vangheluwe. *Multi-Formalism Modelling and Simulation*. PhD thesis, University of Gent, 12 2000.

[154] Hans Vangheluwe, Juan de Lara, and Pieter J. Mosterman. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems) / F. Barros and N. Giambiasi (eds.)*, pages 9–20, 04 2002.

[155] W3C. Web Ontology Language (OWL), 2012. URL `https://www.w3.org/2001/sw/wiki/OWL`. Accessed: 2021-10-15.

[156] Gabriel A. Wainer. *Converting High Level Models into DEVS Modeling and Simulation Applications*, pages 117–156. Springer International Publishing, 1st edition, 05 2019. ISBN 9783030171636. doi: 10.1007/978-3-030-17164-3_7.

[157] Stephan Weißleder and Hartmut Lackner. Top-down and bottom-up approach for model-based testing of product lines. *Electronic Proceedings in Theoretical Computer Science*, 111, 03 2013. doi: 10.4204/EPTCS.111.7.

[158] Tim Weilkiens. *Systems Engineering with SysML/UML*. MK/OMG Press. Morgan Kaufmann Publishers, Amsterdam, 2008. ISBN 9780123742742.

[159] Edmund Widl, Wolfgang Müller, Atiyah Elsheikh, Matthias Hortenhuber, and Peter Palensky. The FMI++ library: A high-level utility package for FMI for model exchange. *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6, 2013.

[160] Edmund Widl, Florian Judex, Katharina Eder, and Peter Palensky. FMI-based co-simulation of hybrid closed-loop control system models. In *Proceedings of the 2015 International Conference on Complex Systems Engineering (ICCSE)*, pages 1–6, 11 2015. doi: 10.1109/ComplexSys.2015.7385981.

[161] Albert Wayne Wymore. *Model-Based Systems Engineering*. CRC Press, Inc., USA, 1st edition, 1993. ISBN 084938012X.

[162] Bernard P. Zeigler. *Theory of Modeling and Simulation*. John Wiley, 1976.

[163] Bernard P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press Professional, Inc., USA, 1984. ISBN 0127784500.

[164] Bernard P. Zeigler and Phillip E. Hammonds. *Modeling and Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*. Academic Press, Inc., USA, 2007. ISBN 9780123725158.

[165] Bernard P. Zeigler and Hessam S. Sarjoughian. *Guide to Modeling and Simulation of Systems of Systems*. Springer Publishing Company, Incorporated, 2013. ISBN 9780857298645.

[166] Bernard P. Zeigler, Cheng-Jye Luh, and Tag-Gon Kim. Model base management for multifacetted systems. *ACM Transactions on Modeling and Computer Simulation*, 1 (3):195–218, July 1991. ISSN 1049-3301. doi: 10.1145/140765.140791.

[167] Bernard P. Zeigler, Tag Gon Kim, and Herbert Prähofer. *Theory of Modeling and Simulation*. Academic Press, Inc., USA, 2nd edition, 2000. ISBN 0127784551.

[168] Bernard P. Zeigler, Chungman Seo, and Doohwan Kim. System Entity Structures for suites of simulation models. *International Journal of Modeling, Simulation, and Scientific Computing*, 04(03):1340006, 2013. doi: 10.1142/S1793962313400060.

[169] Bernard P. Zeigler, Saurabh Mittal, and Mamadou Kaba Traoré. MBSE with/out simulation: State of the art and way forward. *Systems*, 6(4):40, 2018. doi: 10.3390/systems6040040.

[170] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*. Academic Press, Inc., USA, 3rd edition, 2018. ISBN 9780128133705.

[171] Lin Zhang, Bernard Zeigler, and Yuanjun Laili. *Model Engineering for Simulation*. Academic Press, Inc., 01 2019. ISBN 9780128135433. doi: 10.1016/B978-0-12-813543-3.00001-9.

[172] Borut Zupančič, Rihard Karba, and Maja Atanasijevic-Kunc. Continuous systems modelling education - causal or acausal approach? *Proceedings of the ITI 2008 - 30th International Conference on Information Technology Interfaces*, pages 803–808, 06 2008. doi: 10.1109/ITI.2008.4588514.

List of Figures

List of Tables

List of Listings

Appendix

A Formal Discrete Event System Specifications According to Zeigler et al. [167] and Zeigler et al. [170]

In this section the individual elements of the atomic and coupled *Parallel Discrete Event System Specification* (PDEVS) algorithm are listed. The PDEVS algorithm is an extension of the classic *Discrete Event System Specification* (DEVS) algorithm with a focus on the treatment of parallel events. The PDEVS formalism of atomic and coupled discrete event models is outlined in Table A.1.

Table A.1: PDEVS specification of atomic and coupled models based on Zeigler et al. [167] and Zeigler et al. [170].

Atomic model	*Coupled model*
$M = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$	$N = (X, Y, D, M_d \vert d \in D, EIC, EOC, IC)$
where	
$X = \{(p, v) \vert p \in IPorts, v \in X_P\}$: set of input ports and values	$X = \{(p, v) \vert p \in IPorts, v \in X_P\}$: set of input ports and values
$Y = \{(p, v) \vert p \in OPorts, v \in Y_P\}$: set of output ports and values	$Y = \{(p, v) \vert p \in OPorts, v \in Y_P\}$: set of output ports and values
S: set of sequential states	D: set of component names
δ_{ext}: $Q \times X \to S$ external state transition function	M_d: set of PDEVS components
δ_{int}: $S \to S$ internal state transition function	EIC: set of external input couplings
δ_{con}: $S \times X \to S$ confluent state transition function	EOC: set of external output couplings
λ: $S \to Y$ output function	IC: set of internal couplings
ta: $S \to R^+_{0,\infty}$ time advance function	
with	
$Q = \{(s, e) \vert s \in S, 0 \leq e \leq ta(s)\}$ with e elapsed time since the last state transition: set of total states	

B Pruning Design Patterns

As discussed in Section 2.5.2, *Feature Model*s (FMs) are widely used in engineering. In the context of feature modeling, four kinds of features are used: (i) mandatory features (logical AND), (ii) alternative features (logical XOR), (iii) optional features, and (iv) or-features (logical OR). In analogy to the semantics of feature models and mathematical logical expressions, patterns to express these relations with *System Entity Structure*s (SES) are presented here. The patterns in Sections B.1, B.2, and B.7 can be classified as mandatory (logical AND). The patterns in Sections B.3, B.4, B.5, and B.6 can be seen as alternative (logical XOR). For expressing optional features like pattern B.8 and OR features like pattern B.9, an extension of the SES is used. The patterns in Sections B.10, B.11, B.12, and B.13 are combinations of the previous mentioned basic patterns. A modification of pattern B.7 is presented in pattern B.14.

B.1 Aspect Node

An SES tree with a single aspect is the simplest pattern given in Figure B.1. A coupled system *a* consists of an entity *b* and an entity *c*. For model generation aspect nodes need to define the special attribute for couplings, while the leaf nodes have the mb-attribute attached referring to a basic model. Note, that the types of the ports of basic models, which the couplings connect, are left away to keep matters simple. Also note, that the derived *Pruned Entity Structure* (PES) and *Flattened Pruned Entity Structure* (FPES) are identical to the SES. The resulting abstract, simulator-independent model on the right side is given to illustrate what kind of model structure would result from the FPES. In the next sections coupling definitions, the FPES, and the resulting model are not given.

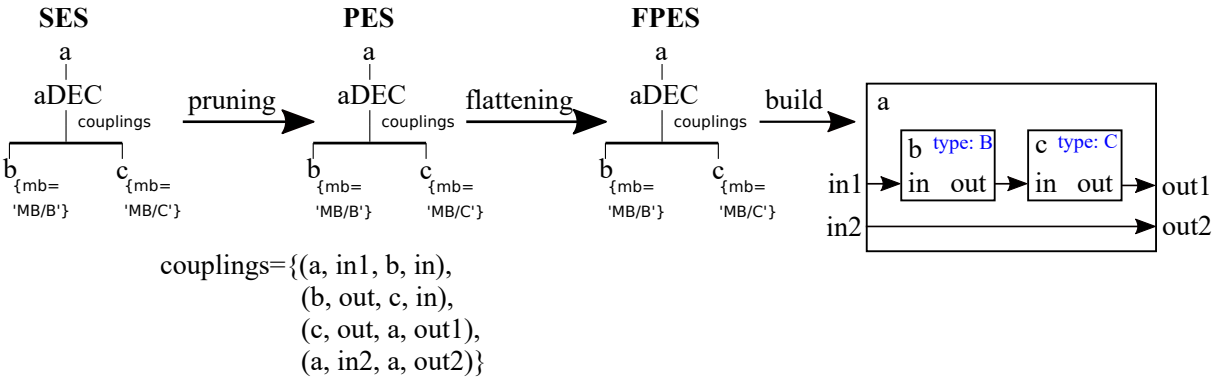


Figure B.1: SES with aspect node, derived PES, FPES, and the corresponding model.

B.2 Multi-Aspect Node

In Figure B.2 a similar case to B.1 is given. The system *as* consists of three children of type *b*. Multi-aspect nodes need to define two special attributes, the attributes *Number of Replications* (NumRep) and couplings. In this pattern the NumRep attribute is filled with a hardcoded three, but the value could also be assigned dynamically by using an

SES variable (SESvar) or *SES function* (SESfcn). When pruning a multi-aspect node the NumRep attribute is evaluated and an aspect node with children of the same type is created. The derived PES is similar to the PES in Figure B.1, except that there is a third brother *b3* and all children of *as* refer to the same model B in the *Model Base* (MB). Children names are generated by appending a number to the entity name *b* to ensure that the resulting siblings fulfill the axiom of valid brothers.

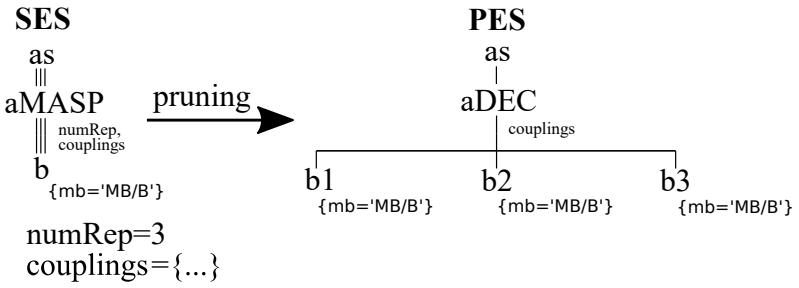


Figure B.2: SES with multi-aspect node and one derived PES.

B.3 Specialization Node

An SES with a specialization node is shown in Figure B.3. Based on the specrule exactly one child needs to be selected to construct a valid variant. Thus, after pruning the resulting system can either be of type *b* OR of type *c*. The father inherits the attributes of the child.

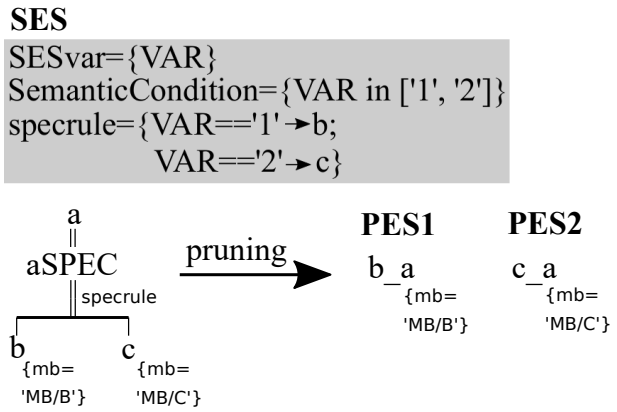


Figure B.3: SES with specialization node and both possible PES.

B.4 Aspect Siblings

If two or more aspect nodes are on the same hierarchy level, exactly one of them has to be selected by evaluating the aspect rules of the aspect brothers. This is presented in Figure B.4. The system *a* consists either of *b* and *c* or of *d* and *e*. The aspect rules one and two define which branch is selected during pruning.

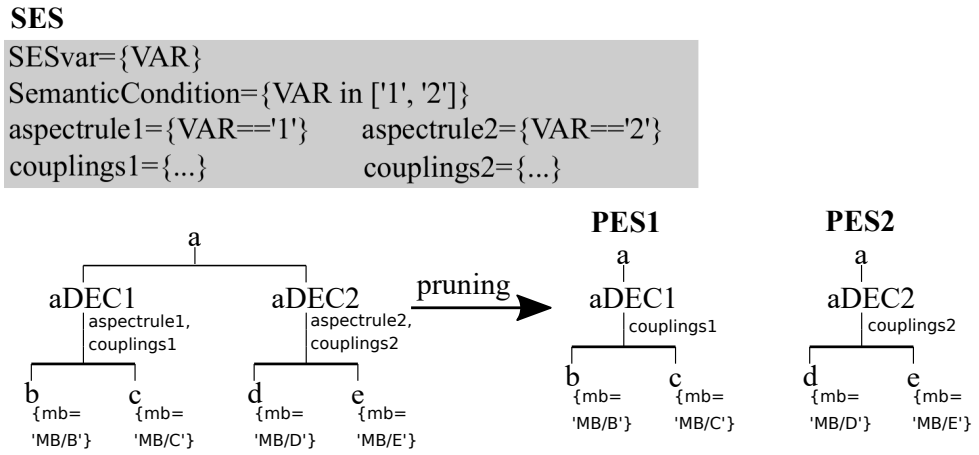


Figure B.4: SES with aspect siblings and both possible PES.

B.5 Multi-Aspect Siblings

In the pattern shown in Figure B.5 two multi-aspect nodes are on the same layer. In the first pruning step the multi-aspect nodes are resolved leading to two aspect nodes. After this step, the resulting intermediate PES can be finally resolved as in Section B.4 for aspect siblings previously described. The system *a* consists either of *b1*, *b2*, and *b3* or of *c1* and *c2* depending on which child is selected based on the aspect rules.

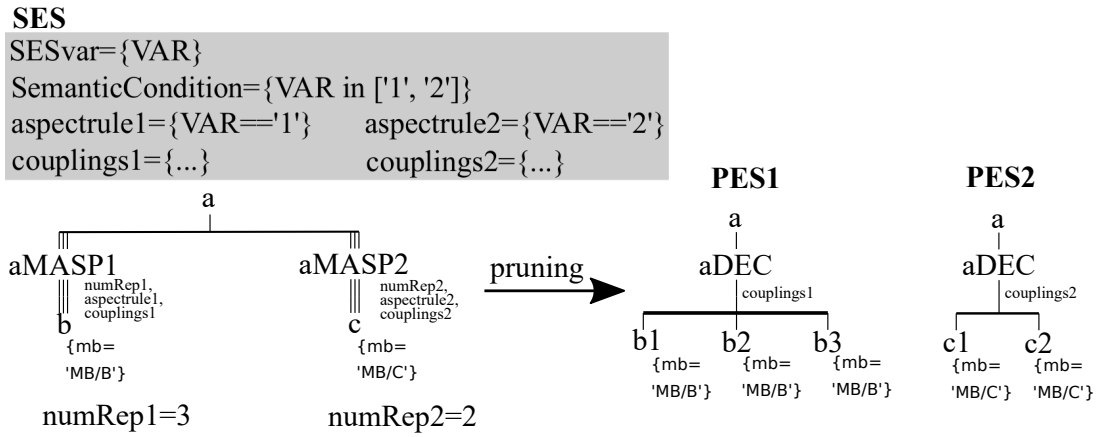


Figure B.5: SES with multi-aspect siblings and possible PES.

B.6 Aspect and Multi-Aspect Siblings

If there are more than one aspect nodes and multi-aspect nodes on the same hierarchy level, the behavior is like aspect siblings after the multi-aspect node is resolved. The pattern is shown in Figure B.6. After pruning the system is built up as before in Section B.5.

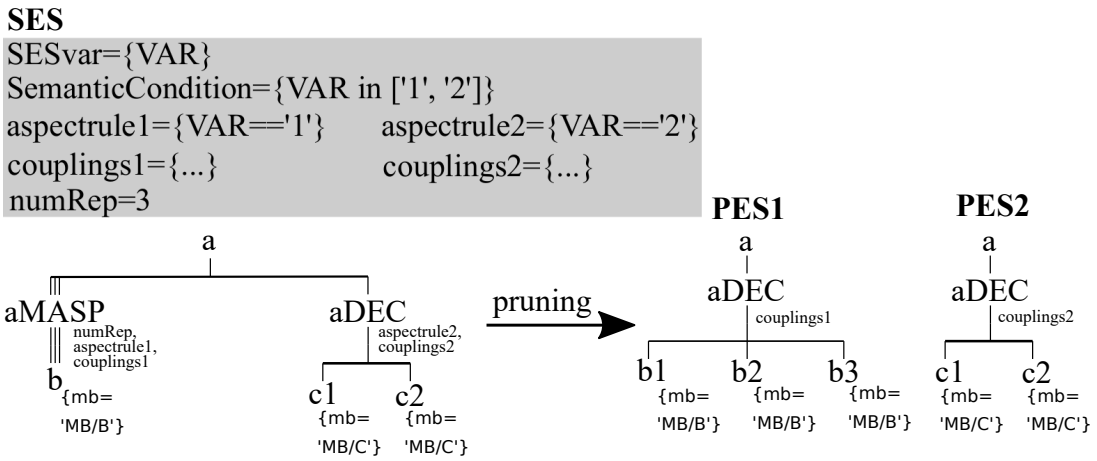


Figure B.6: SES with aspect and multi-aspect siblings and possible PES.

B.7 Specialization Siblings

If two or more specialization nodes are on the same hierarchy level, they will all be evaluated. For the pattern depicted in Figure B.7, this means that *a* will specialize into one of *b* or *c* AND into one of *d* or *e*. Which child of the specialization is taken depends on the values of the SESvars and the specrules. If, for example, VAR1 is set to 1 and VAR2 is set to 2, at specialization *aSPEC1* the left child *b* is selected and at specialization *aSPEC2* the right child *e* is selected. During pruning it depends on the pruning algorithm which of the two specializations is evaluated first. In this pattern it is assumed that the left specialization node *aSPEC1* is evaluated first. The evaluation order influences what is the resulting value of the mb-attribute since, according to the inheritance axiom, attributes with the same name are overwritten in the parent node.

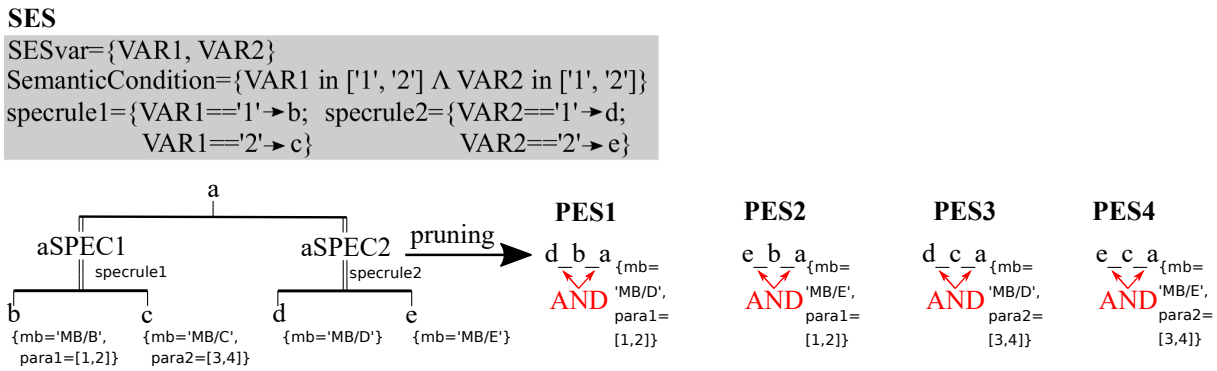


Figure B.7: SES with specialization siblings and all four possible PES.

B.8 The NONE Element

The NONE element is an extension of the SES. Nodes can have the name NONE. A NONE element for a leaf entity node means that if the NONE branch is selected during pruning, the entity is not included at all. In the pattern shown in Figure B.8 the specialization has

a child which is a NONE element. Hence, this SES can evaluate to NONE during pruning based on the specrule. The system *a* can either be of type *b* or not existent at all.

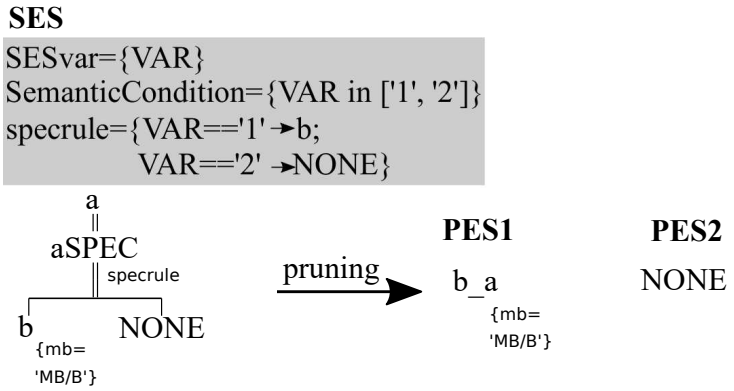


Figure B.8: SES with the NONE element at a specialization.

B.9 Express OR in the SES

The pattern in Figure B.9 shows an aspect node whose children are followed by specializations. Each specialization contains a NONE element (see Section B.8) as one child. For expressing a logical OR, at least one specialization has to evaluate to a node not being NONE, as coded in the semantic condition. By defining the specrules reasonably, the user has to ensure that this is guaranteed and the semantic conditions are met. This pattern is composed of the pattern in Section B.1 in combination with the pattern in Section B.8. After pruning, the system *a* consists of *b* and *c*. System *b* in turn is of type *bs* or not existent while *c* is of type *cs* or not existent. Couplings have to be justified when the tree changes by evaluating nodes. Since the couplings are defined at the aspect node *aDEC*, it is obvious that there is a need for the possibility to define variable couplings. Variable couplings can be defined by SESfcns.

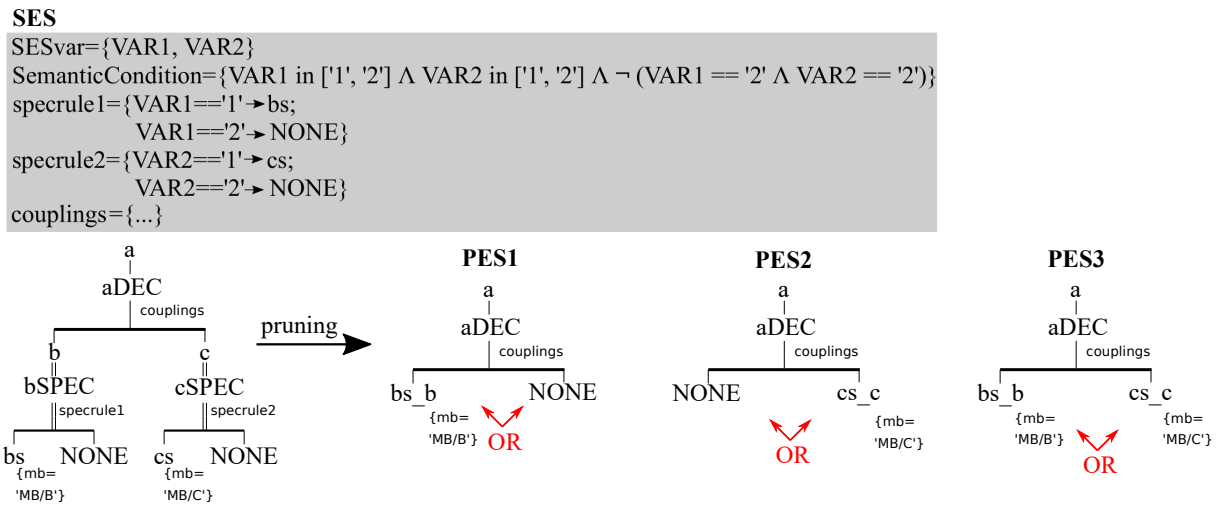


Figure B.9: SES to express OR.

B.10 Two Specialization Nodes in One Path

During pruning specialization nodes inherit the attributes of the selected child and append them to the father's attributes as described previously. In Figure B.10, there are two specialization nodes in one path. Additionally, the NONE element is used. This shall clarify the axiom for attribute inheritance. During pruning, first *aSPEC* is evaluated. In case the child *c* is selected, another decision for child *d* or child *e* is taken. In that case the attributes of the child node of *cSPEC* are inherited to the first node.

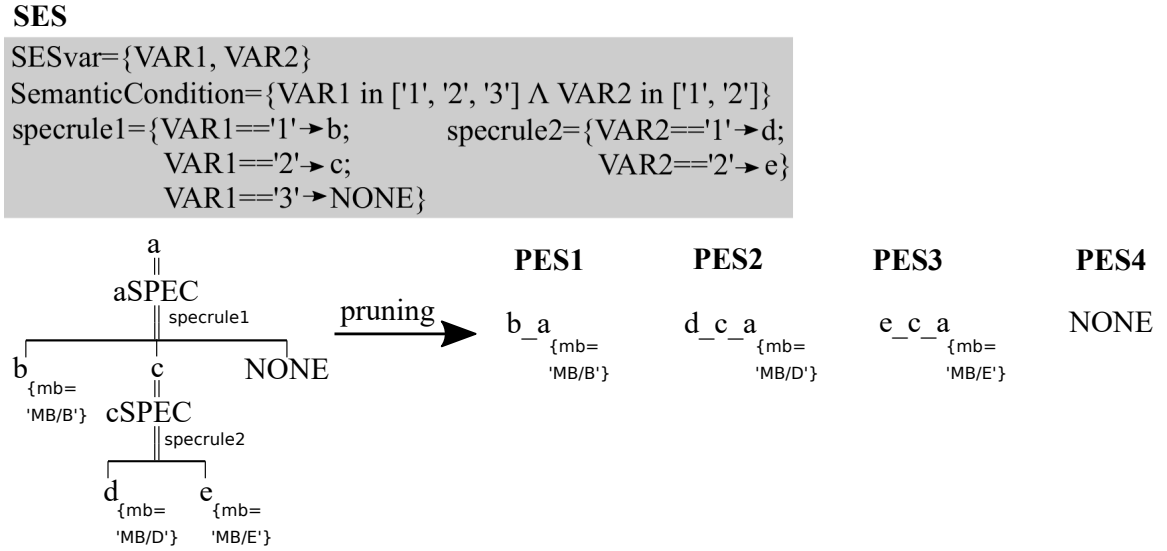


Figure B.10: Two specialization nodes in one path.

B.11 Specialization with Succeeding Aspect

Figure B.11 depicts a specialization node with a succeeding aspect node. This pattern is a combination of a specialization node followed by a single aspect node. The system *a* can be of type *b* or *c*, while *b* can be decomposed in *d* and *e*.

B.12 Specialization and Aspect Siblings

When aspect nodes and specialization nodes are brothers, the specialization node has to be resolved first during pruning. If, additionally, an aspect node is below the specialization node, during pruning two aspect nodes will become siblings. Since in this case the occurrence of aspect siblings is not known until the first pruning step, aspect rules could not be formulated beforehand. In order to tackle this the *priority attribute* for aspect nodes was introduced. Throughout the whole SES aspect nodes get a unique number. If during pruning aspect nodes become brothers, a decision as to which to choose can be found. Nodes with higher priority numbers are priorized over those with lower numbers. In Figure B.12, a pattern is given. The system *a* consists of *b* and *c*, if it is of type *d*. If it is of type *e*, it can consist of *b* and *c* or *f* and *g*. In case that the system is of type *e*, a decision as to which decomposition to take is made by the priority attribute.

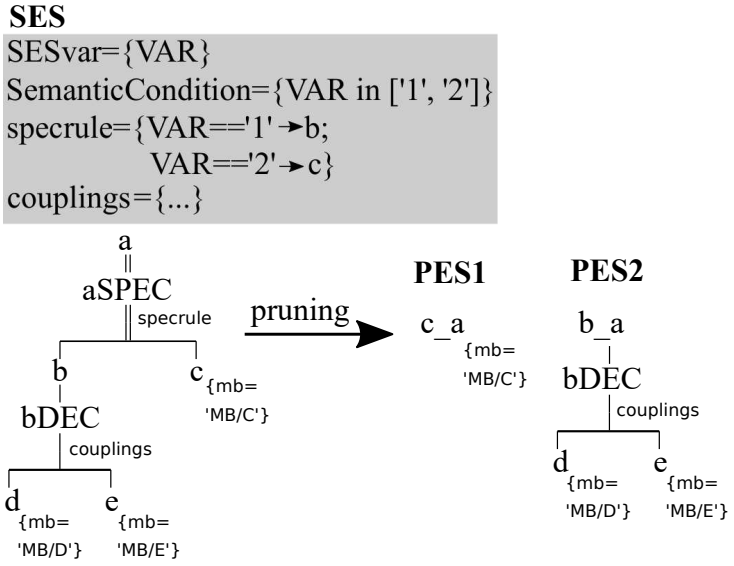


Figure B.11: Specialization with succeeding aspect.

B.13 Several Multi-Aspects in a Path

If a multi-aspect node is followed by a second or even more multi-aspects, complexity of resulting structures grows considerably. During pruning compliance with the SES axioms has to be ensured and therefore some renaming operations are necessary. Figure B.13 depicts two successive multi-aspects where the NumRep variable of *bMASP* defines a multiset. Renaming of *b* to *b1* and *b2*, as necessary for children of multi-aspects and explained in the pattern in Section B.2, results in renaming of *bMASP* to *b1MASP* and *b2MASP*. Generally, one can say that renaming an entity node always calls for renaming the following descriptive node as well. The first pruning step resolves the multi-aspect *aMASP* to an aspect *aDEC* with two child nodes of type multi-aspect. In a second step the child nodes are resolved to aspect nodes as well. Since the NumRep variable of *b1MASP* and *b2MASP* was set via the multiset, variable couplings are necessary.

This example is intended to increase the understanding of the developed pruning algorithm in Section 3.2.2. However, an enhanced approach for automatic pruning of hierarchical multi-aspect nodes is introduced in Section 3.2.4.

B.14 Selection Constraints

A modification of the specialization siblings in Section B.7 is shown in Figure B.14. With this pattern the use of selection constraints and semantic conditions to limit the possible valid structures is pointed out. In FMs the corresponding constructs are known as *require* and *exclude*. The dotted line from leaf node *c* to node *d* in Figure B.14 depicts a selection constraint, which means that if *c* is selected at node *a1SPEC*, *d* needs to be chosen at *a2SPEC*. The resulting PES is PES3. The current value of *VAR2* does not matter for the selection if *VAR1* has the value 2. Another way to control possible variants are the semantic conditions. The semantic conditions for value combinations of *VAR1* and *VAR2* interdict the selection of *d* if *b* is already selected.

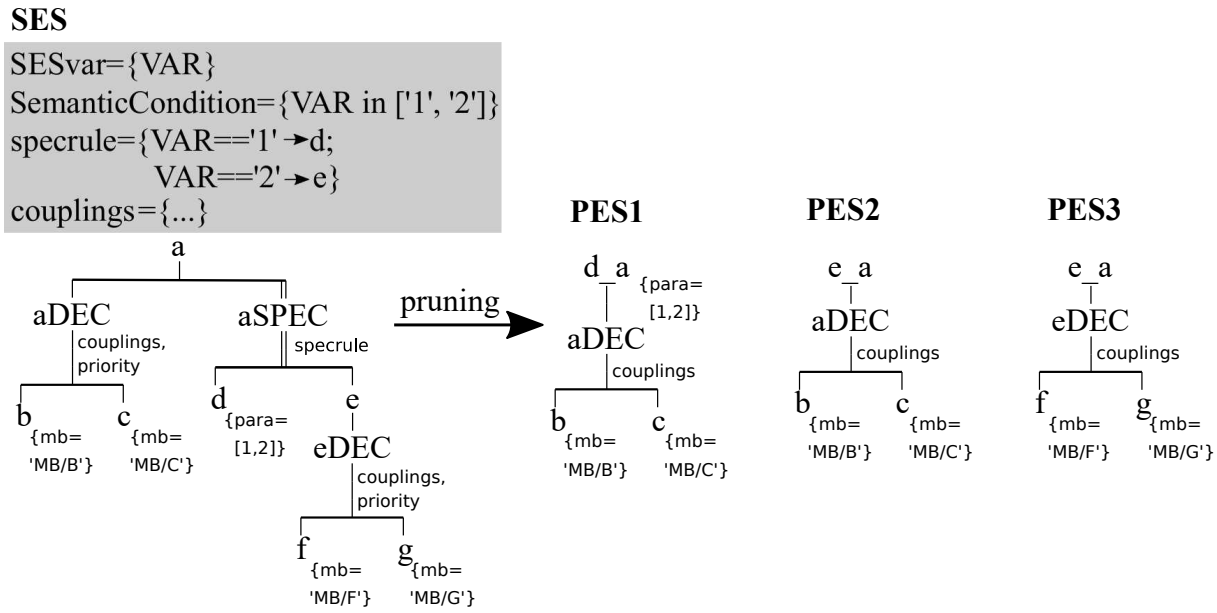


Figure B.12: Specialization and aspect siblings.

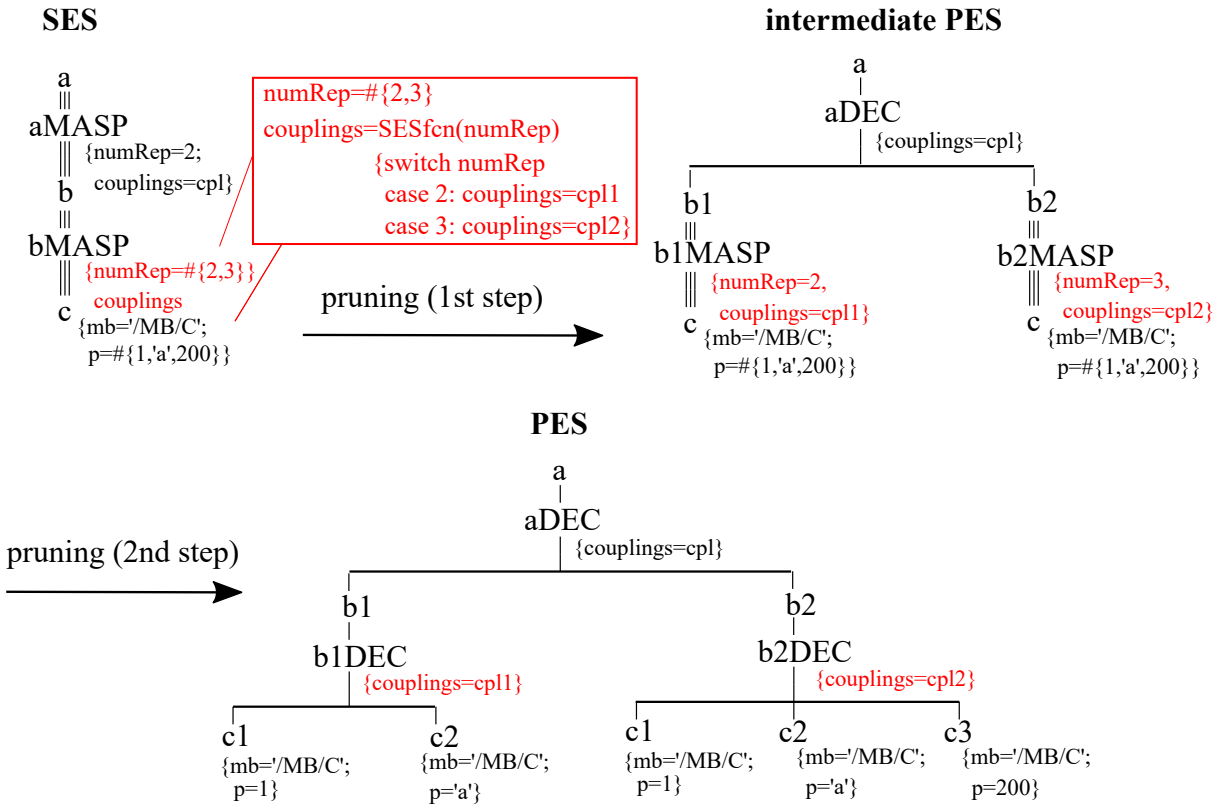


Figure B.13: Successive multi-aspects with variable number of replications.

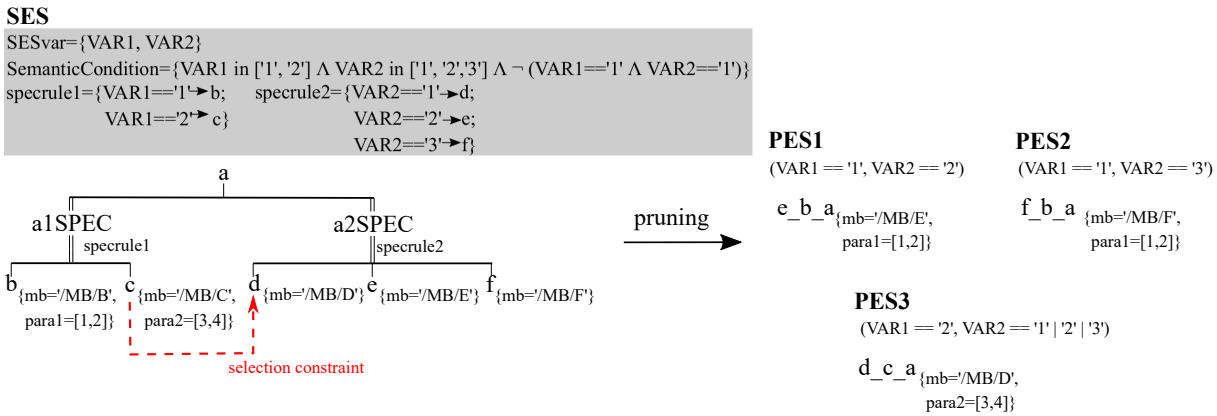


Figure B.14: Variant restriction with selection constraints and semantic conditions.

C Couplings of the Feedback Control System

Section 3.2.3 discusses the SES, the PES, and the FPES with their respective couplings. While the couplings of the SES coding system configurations of a feedback control system are given completely via a coupling list and an SESfcn, only selected couplings are shown for the derived PES and the FPES. At this point, complete lists of the couplings of the PES and the FPES are presented.

In Figure 3.4, the PES number 1 shows the variant of the feedback control system without feedforward control. It only has the couplings *cplg1*. Furthermore, the PES corresponds to the FPES, so that no further adaptation is necessary. Table C.1 shows these couplings.

Table C.1: The couplings of the PES number 1 in Figure 3.4.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
cplg1					
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
procUnitSys	y	SPR	addDist	u2	SPR
addDist	y	SPR	feedbackSys	u2	SPR
sourceDist	y	SPR	tfDist	u	SPR
tfDist	y	SPR	addDist	u1	SPR
ctrlPIDSys	y	SPR	procUnitSys	u	SPR

In Figure 3.4, the PES number 2 shows the variant of the feedback control system with feedforward control. This variant has the couplings *cplg1* and *cplg2*. These couplings are given in Table C.2. When flattening, inner nodes are removed and the couplings are adapted accordingly and united in the only *cplg* attribute as presented in Figure 3.5. These adapted couplings of the FPES are presented in Table C.3.

Table C.2: The couplings of the PES number 2 in Figure 3.4.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
cplg1					
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
procUnitSys	y	SPR	addDist	u2	SPR
addDist	y	SPR	feedbackSys	u2	SPR
sourceDist	y	SPR	tfDist	u	SPR
tfDist	y	SPR	addDist	u1	SPR
sourceDist	y	SPR	fc_feedforwardCtrl	u1	SPR
ctrlPIDSys	y	SPR	fc_feedforwardCtrl	u2	SPR
fc_feedforwardCtrl	y	SPR	procUnitSys	u	SPR
cplg2					
fc_feedforwardCtrl	u1	SPR	tfFeedforward	u	SPR
tfFeedforward	y	SPR	addFeedforward	u1	SPR
fc_feedforwardCtrl	u2	SPR	addFeedforward	u2	SPR
addFeedforward	y	SPR	fc_feedforwardCtrl	y	SPR

Table C.3: The couplings of the FPES in Figure 3.5.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
cplg					
sourceSys	y	SPR	feedbackSys	u1	SPR
feedbackSys	y	SPR	ctrlPIDSys	u	SPR
procUnitSys	y	SPR	addDist	u2	SPR
addDist	y	SPR	feedbackSys	u2	SPR
sourceDist	y	SPR	tfDist	u	SPR
tfDist	y	SPR	addDist	u1	SPR
addFeedforward	y	SPR	procUnitSys	u	SPR
sourceDist	y	SPR	tfFeedforward	u	SPR
tfFeedforward	y	SPR	addFeedforward	u1	SPR
ctrlPIDSys	y	SPR	addFeedforward	u2	SPR

D XML Structure of the SES of the Feedback Control System

In Section 2.5.3 the SES/MB approach in the context of *Model-Driven Engineering* (MDE) and *Model-Driven Architecture* (MDA) is discussed. A key requirement is the interchangeability of models according to the MDA. In Zeigler and Hammonds [164] and Zeigler and Sarjoughian [165] an *Extensible Markup Language* (XML) structure is introduced for the classic SES/MB framework. The extensions discussed in 3.2 are not supported. Therefore a new XML representation of this extended SES is developed. The XML representation of the SES coding system configurations of the feedback control system is shown in Listing D.1.

```
<?xml version="1.0" ?>
<SESwithSettings name="SES" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance">
    <globals>
        <sespes comment="" value="ses"/>
        <sesvars>
            <sesvar comment="" name="feedforward" value="0"/>
        </sesvars>
        <semcons>
            <semcon result="T" value="feedforward in [0,1]"/>
        </semcons>
        <selcons/>
        <sesfcns>
            <sesfcn fcn="def cplfcn(feedforward, children):
            #children[0] is feedforwardCtrl
            #children[1] is sourceSys
            #children[2] is feedbackSys
            #children[3] is ctrlPIDSys
            #children[4] is procUnitSys
            #children[5] is sourceDist
            #children[6] is tfDist
            #children[7] is addDist

            cplg = []

            #fixed couplings
            cplg.append([children[1],&quot;y / SPR&quot;,children[2],&quot;u1 / SPR&quot;,&quot;&quot;])
            cplg.append([children[2],&quot;y / SPR&quot;,children[3],&quot;u / SPR&quot;,&quot;&quot;])
            cplg.append([children[4],&quot;y / SPR&quot;,children[7],&quot;u2 / SPR&quot;,&quot;&quot;])
            cplg.append([children[7],&quot;y / SPR&quot;,children[2],&quot;u2 / SPR&quot;,&quot;&quot;])
            cplg.append([children[5],&quot;y / SPR&quot;,children[6],&quot;u / SPR&quot;,&quot;&quot;])
            cplg.append([children[6],&quot;y / SPR&quot;,children[7],&quot;u1 / SPR&quot;,&quot;&quot;])

            #variable couplings
            if feedforward==0:
            cplg.append([children[3],&quot;y / SPR&quot;,children[4],&quot;u / SPR&quot;,&quot;&quot;])
            elif feedforward==1:
            cplg.append([children[5],&quot;y / SPR&quot;,children[0],&quot;u1 / SPR&quot;,&quot;&quot;])
            cplg.append([children[3],&quot;y / SPR&quot;,children[0],&quot;u2 / SPR&quot;,&quot;&quot;])
            cplg.append([children[0],&quot;y / SPR&quot;,children[4],&quot;u / SPR&quot;,&quot;&quot;])

            #return
            return cplg
            " fcnname="cplfcn" language="Python"/>
        </sesfcns>
    </globals>
    <tree>
```

```
47              <node comment="" name="ctrlSys" type="entity">
48                  <node comment="" name="ctrlSysDEC" type="aspect">
49                      <aspr comment="" condition="" result=""/>
50                      <prio value="1"/>
51                      <cplg comment="" cplgfcn="cplfcn(feedforward,CHILDREN)" sinknode="" sinkport="" sinktype="" sourcenode=""
                             sourceport="" sourcetype=""/>
52                      <node comment="" name="feedforwardCtrl" type="entity">
53                          <node comment="" name="feedforwardCtrlSPEC" type="specialization">
54                              <specr comment="" condition="feedforward==1" fornode="fc" result="F"/>
55                              <specr comment="" condition="feedforward==0" fornode="NONE" result="T"/>
56                              <node comment="" name="fc" type="entity">
57                                  <node comment="" name="fcDEC" type="aspect">
58                                      <aspr comment="" condition="" result=""/>
59                                      <prio value="1"/>
60                                      <cplg comment="" cplgfcn="" sinknode="tfFeedforward" sinkport="u" sinktype="SPR" sourcenode="
                                             fc" sourceport="u1" sourcetype="SPR"/>
61                                      <cplg comment="" cplgfcn="" sinknode="addFeedforward" sinkport="u1" sinktype="SPR" sourcenode=
                                             "tfFeedforward" sourceport="y" sourcetype="SPR"/>
62                                      <cplg comment="" cplgfcn="" sinknode="addFeedforward" sinkport="u2" sinktype="SPR" sourcenode=
                                             "fc" sourceport="u2" sourcetype="SPR"/>
63                                      <cplg comment="" cplgfcn="" sinknode="fc" sinkport="y" sinktype="SPR" sourcenode="
                                             addFeedforward" sourceport="y" sourcetype="SPR"/>
64                                      <node comment="" name="tfFeedforward" type="entity">
65                                          <attr comment="" name="mb" value="'MB/TransferFunction'" varfun=""/>
66                                          <attr comment="" name="b" value="'{20,1}'" varfun=""/>
67                                          <attr comment="" name="a" value="'{10,1}'" varfun=""/>
68                                      </node>
69                                      <node comment="" name="addFeedforward" type="entity">
70                                          <attr comment="" name="mb" value="'MB/Add'" varfun=""/>
71                                          <attr comment="" name="k1" value="-1" varfun=""/>
72                                      </node>
73                                  </node>
74                              </node>
75                              <node comment="" name="NONE" type="entity"/>
76                          </node>
77                      </node>
78                      <node comment="" name="sourceSys" type="entity">
79                          <attr comment="" name="mb" value="'MB/Constant'" varfun=""/>
80                          <attr comment="" name="k" value="0" varfun=""/>
81                      </node>
82                      <node comment="" name="feedbackSys" type="entity">
83                          <attr comment="" name="mb" value="'MB/Feedback'" varfun=""/>
84                      </node>
85                      <node comment="" name="ctrlPIDSys" type="entity">
86                          <attr comment="" name="mb" value="'MB/PID'" varfun=""/>
87                          <attr comment="" name="k" value="1" varfun=""/>
88                          <attr comment="" name="Ti" value="1" varfun=""/>
```

```
89                          <attr comment="" name="Td" value="0" varfun=""/>
90                      </node>
91                      <node comment="" name="procUnitSys" type="entity">
92                          <attr comment="" name="mb" value="'MB/TransferFunction'" varfun=""/>
93                          <attr comment="" name="b" value="'{1}'" varfun=""/>
94                          <attr comment="" name="a" value="'{20,1}'" varfun=""/>
95                      </node>
96                      <node comment="" name="sourceDist" type="entity">
97                          <attr comment="" name="mb" value="'MB/Step'" varfun=""/>
98                          <attr comment="" name="startTime" value="0.5" varfun=""/>
99                      </node>
100                     <node comment="" name="tfDist" type="entity">
101                         <attr comment="" name="mb" value="'MB/TransferFunction'" varfun=""/>
102                         <attr comment="" name="b" value="'{1}'" varfun=""/>
103                         <attr comment="" name="a" value="'{10,1}'" varfun=""/>
104                     </node>
105                     <node comment="" name="addDist" type="entity">
106                         <attr comment="" name="mb" value="'MB/Add'" varfun=""/>
107                     </node>
108                 </node>
109             </node>
110         </tree>
111     </SESwithSettings>
```

Listing D.1: XML representation of the SES coding system configurations of the feedback control system.

E Excerpt of a Possible Experiment Control Script for the Feedback Control Example

The *Experiment Control* (EC) SESEcPy is introduced as template script in Section 4.4.1. The concrete implementation of the SESEcPy for the feedback control example is shown here based on the experiment specific steps for the experimentation specified in Section 4.4.4.

As described, SESEcPy is structured in a general and an experiment specific part. In the experiment specific part the experiment run is coded by functions, such as the SES file and initial or reactively derived SESvars. In the general part the control and calling order of the components of the SES/MB-based architecture are coded by a function. Furthermore, a loop is required in which the experiment steps are processed. This is implemented in a *main* script of SESEcPy. An excerpt of the main script of SESEcPy is shown in Listing E.1. At first the initial experiment specific SES structure needs to be loaded as shown in the function *initialSettings()* in Listing E.2. In the loop the experiment-specific and general functions are called alternately. In the experiment specific function *nextState(results)* in Listing E.2 a current setting of SESvars or an termination of the experiment and extensions of the *config file* are calculated. On the first run *results* is empty, but in following simulation runs existing simulation results can be taken into account. In the function *generalExperimentation(SESfile, SESvar, addConfig)* in Listing E.3 the software tools with its respective methods pruning, flattening, and build are called. In case a method fails, no result is returned to the main script and the experiment is terminated.

```
...
SESfile = experimentSpecific.initialSettings()
stop = 0
results = ""
#main loop until stop==1
while stop == 0:
    SESvar, stop, addConfig = experimentSpecific.nextState(results)
    if stop == 0:
        results = general.generalExperimentation(SESfile, SESvar, addConfig)
...
```

Listing E.1: Excerpt of the *main* script of SESEcPy.

```
function initialSettings()
    SESfile = ...
    ...
    return SESfile

function nextState(results)
    stop = 0
    SESvar = [feedforward=0, mysim=<simulator>, ...]
    #on the first call, results are empty
    if results != "":
        #analyze results
        #check if control goals are met
        if goals are not met:
            SESvar = ...
        else:
            stop = 1
    addConfig = [starttime, finaltime, solver, ...]
    return [SESvar, stop, addConfig]
```

Listing E.2: Excerpt of the *experiment specific* functions of SESEcPy.

```
function generalExperimentation(SESfile, SESvar, addConfig)
    #SESToPy: prune
    PESfile = SESToPy("prune", SESvar, SESfile)
    if PESfile is derived:
        #SESToPy: flatten
        FPESfile = SESToPy("flatten", PESfile)
        if FPESfile is derived:
            #SESMoPy: build
            smHandle = SESMoPy(FPESfile)
            if model created:
                extend configuration file in smHandle with addConfig
                #SESEuPy: simulate
                results = SESEuPy(smHandle)
                return results
            else:
                #return "" and thus stop the experiment
                return ""
        else:
            #return "" and thus stop the experiment
            return ""
    else:
        #return "" and thus stop the experiment
        return ""
```

Listing E.3: Excerpt of the *general* function of SESEcPy.

F Overview of Model Generation with SESMoPy

In Section 4.4.5 model generation using the software SESMoPy is discussed. An activity diagram of SESMoPy's model generation process is presented in this section to provide the overview. It is depicted in Figures F.1 and F.2. The latter figure shows model generation using the *Functional Mock-up Interface* (FMI).

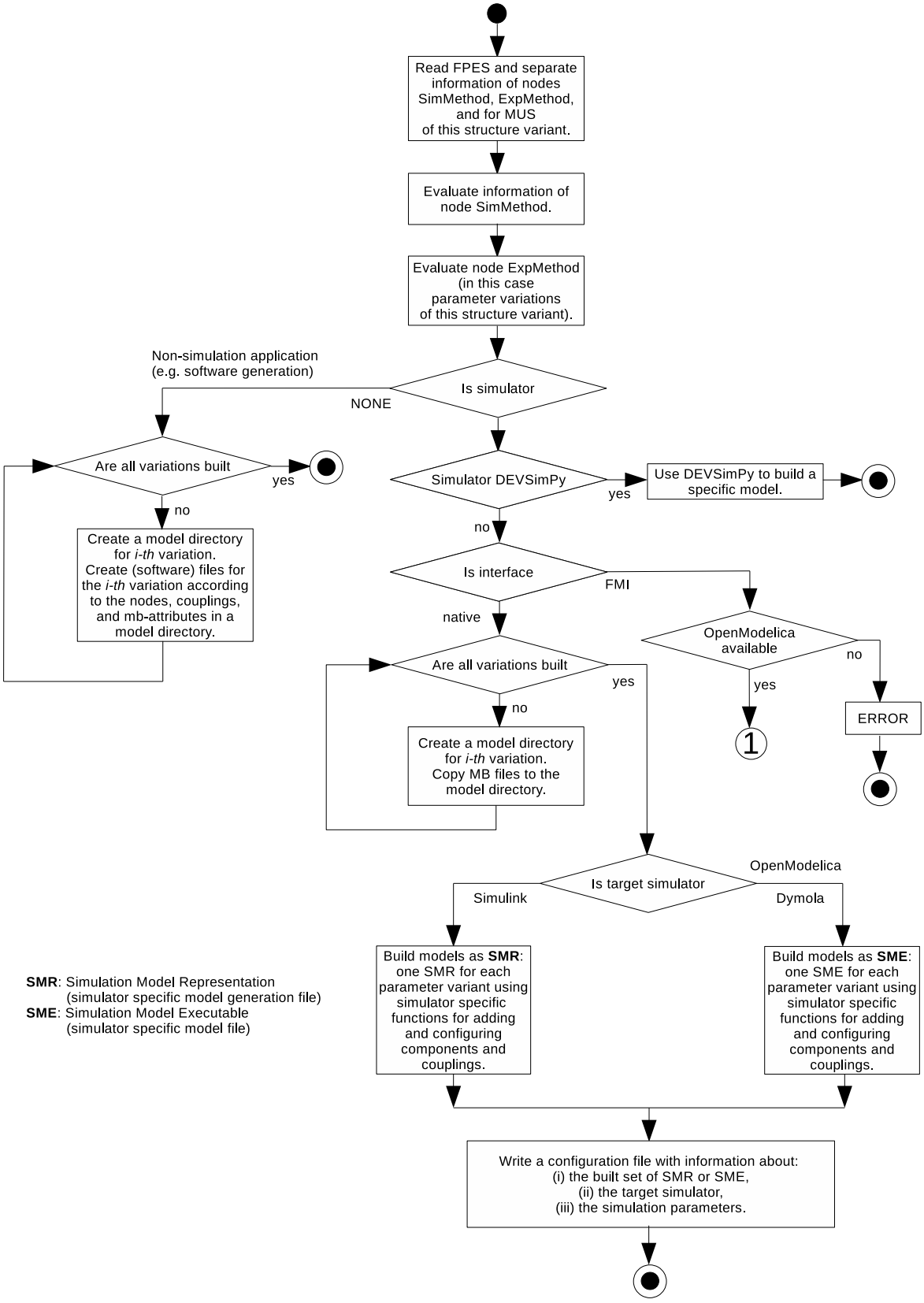


Figure F.1: Steps for model generation with SESMoPy.

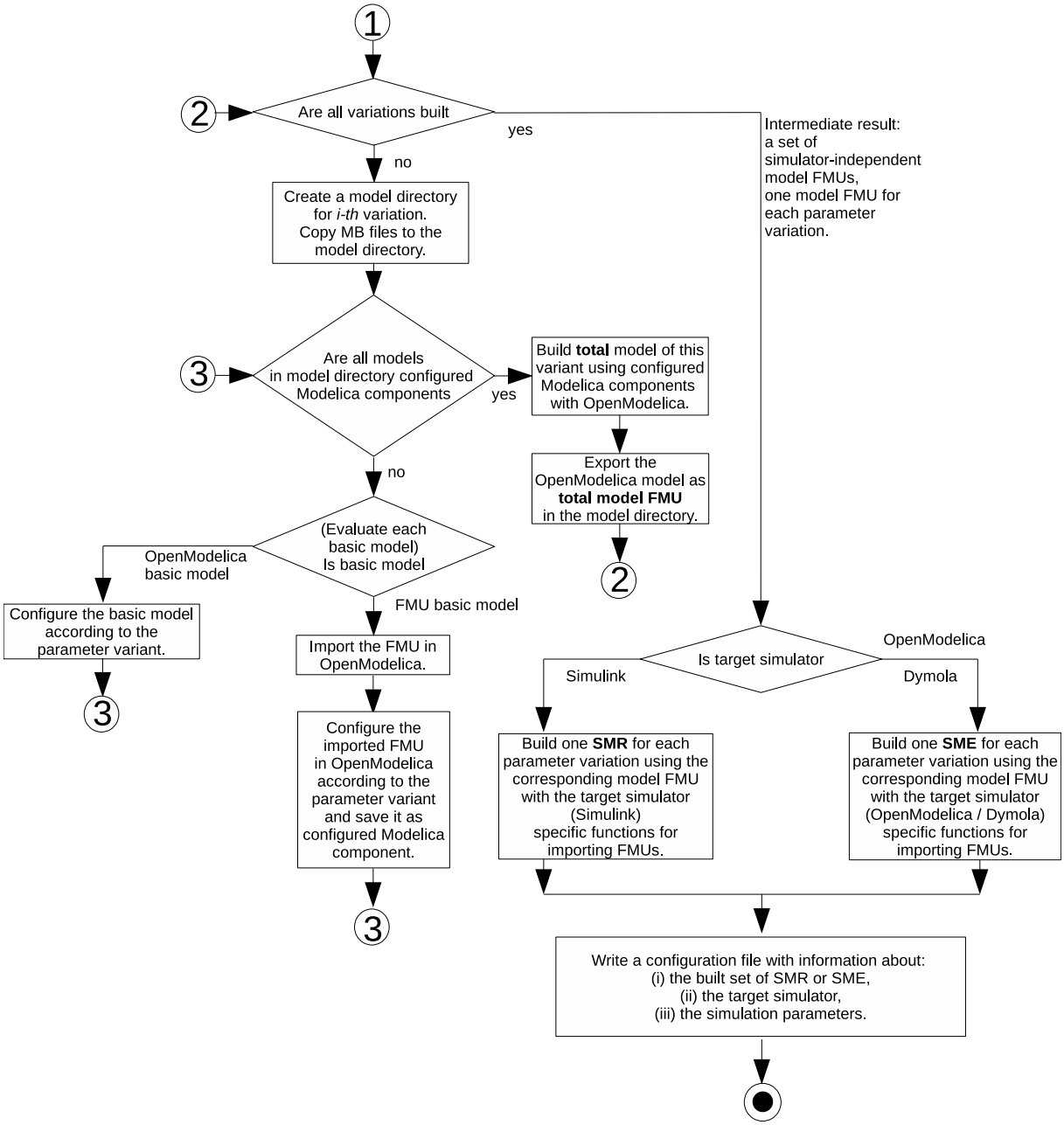


Figure F.2: Steps for model generation with SESMoPy using FMI.

G Excerpt of a Simulation Model Representation and the Resulting Simulation Model Executable for the Feedback Control Example

Listing G.1 shows a Simulink *Simulation Model Representation* (SMR) for *native* model generation. It is an excerpt of the SMR specifying the feedback control system. The SMR specifies the respective MATLAB commands for the model generation, execution, and return values in an M-script. It is generated for MATLAB R2018a.

For model generation basic models are taken from the MB and configured according to the information from the PES or FPES. A Simulink MB for native model generation is shown in Figure 4.6. All basic models are wrapped in submodels for unification of ports and names. For the configuration of basic models an additional function was introduced. It is set as *InitFcn* in the SMR, which is executed on model generation. The *InitFcn* configures blocks with configuration information. An example for this is the configuration information *sourceSys_k=0* for block *sourceSys* in line 25. A template of this additional function was presented in Listing 4.1. After the blocks the couplings are set. In Section 4.4.6 the extension of the SMR by the *Execution Unit* (EU) SESEuPy is described. The SMR is extended with output ports and their respective couplings for variables of interest. Furthermore, the command for simulation execution with its configuration is specified. Finally, simulation results are collected and stored.

On execution in MATLAB an executable Simulink *Simulation Model Executable* (SME) is generated. Figure G.1 shows the respective SME.

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %generate Simulink model
3
4  load_system('simulink');
5  h = new_system('Simulink_Feedback_Model_SMR');
6  open_system(h);
7
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9  %set the InitFcn
10
11 %text for the InitFcn of the model
12 initText = '';
13 %read additional function
14 initText =[initText, fileread('setParameters.m'), newline, newline];
15 %write the contents in InitFcn of the model, it sets parameters of the model
16 set_param(h, 'InitFcn', initText);
17
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19 %add blocks according to the PES or FPES
20
21 %block sourceSys
22 %add the Simulink block from the MB and rename it according to the SES
23 h = add_block('MB/Constant', 'Simulink_Feedback_Model_SMR/sourceSys');
24 %variables for the block -> applied to the block in the InitFcn
25 sourceSys_k = '0';
26
27 %block feedbackSys
28 %add the Simulink block from the MB and rename it according to the SES
29 h = add_block('MB/Feedback', 'Simulink_Feedback_Model_SMR/feedbackSys');
30
31 %add more needed blocks from the Simulink MB according to the PES or FPES
32 ...
33
34 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35 %add couplings according to the PES or FPES
```

```
36
37 %coupling between the blocks sourceSys and feedbackSys
38 %get the port handles of the source block
39 phFrom = get_param('Simulink_Feedback_Model_SMR/sourceSys','PortHandles');
40 %get the port handles of the sink block
41 phTo = get_param('Simulink_Feedback_Model_SMR/feedbackSys','PortHandles');
42 %find port number of the output port (pno)
43 pno = ...
44 %find port number of the input port (pni)
45 pni = ...
46 %create coupling between the output port (type: Outport)
47 %   and input port (type: Inport)
48 add_line('Simulink_Feedback_Model_SMR', phFrom.Outport(pno), ...
49 phTo.Inport(pni), 'autorouting', 'on');
50
51 %add more needed couplings according to the PES or FPES
52 ...
53
54 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55 %Simulink specific extensions added by the EU
56 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
57 %add output blocks for variables of interest
58
59 %add output blocks and their coupling for variables of interest
60 %Simulink_OutBlock is an MB implemented in the EU
61 h = add_block('Simulink_OutBlock/Out', 'Simulink_Feedback_Model_SMR/sourceSys_y');
62 %set the number of the output block
63 set_param(h, 'Port', '1');
64 %get the port handles of the source block
65 phFrom = get_param('Simulink_Feedback_Model_SMR/sourceSys','PortHandles');
66 %get the port handles of the sink block
67 phTo = get_param('Simulink_Feedback_Model_SMR/sourceSys_y','PortHandles');
68 %find port number of the output port (pno)
69 pno = ...
70 %set the port number of the input port (see above)
71 pni = 1;
72 %create coupling between the output port (type: Outport)
73 %   and input port (type: Inport)
74 add_line('Simulink_Feedback_Model_SMR', phFrom.Outport(pno), ...
75 phTo.Inport(pni), 'autorouting', 'on');
76
77 %add more output blocks for variables of interest
78 ...
79
80 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81 %add and parameterize simulation execution
82
83 %Simulator run
84 simout = sim('Simulink_Feedback_Model_SMR', 'StartTime', '0', ...
85 'Solver', 'ode45', 'StopTime', '50', 'MaxStep', '0.1', ...
86 'SaveFormat', 'StructureWithTime');
87
88 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89 %get the simulation results and write them in a CSV file
90 t = ...
91 y = ...
92 simdata = {...};
93 dlmwrite('Simulink_Feedback_Model_SMR.csv',simdata,'-append');
```

Listing G.1: Excerpt of a native SMR of the feedback control system for MATLAB/Simulink extended by the EU.

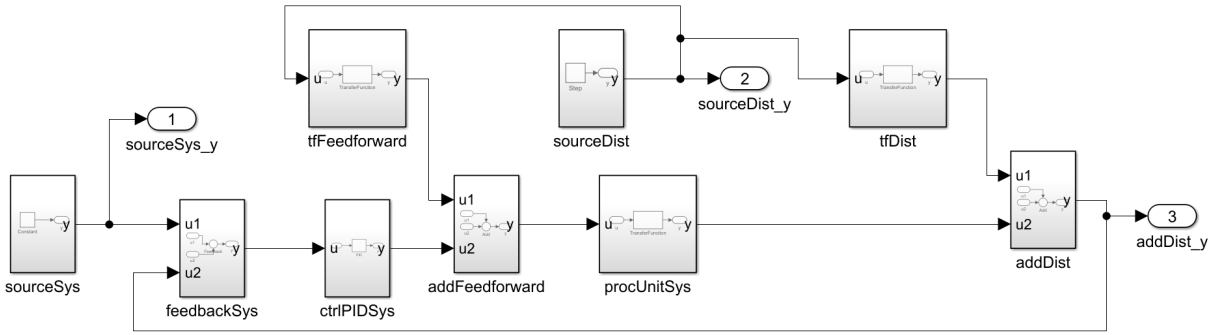


Figure G.1: Native Simulink model of the feedback control system built from the SMR in Listing G.1.

H Variants of the Watershed Example

The watershed example was introduced in Section 5.1. The concepts of abstraction hierarchies and temporal granularity were introduced. However, only essential parts of the SES were shown and no couplings were presented. Figure H.1 shows the full SES describing the watershed system and in the next paragraphs it is discussed.

Next to the SESvar *WSLevel* introduced in Section 5.1 three additional SESvars are needed. These are *MntLevel*, *BasinLevel*, and *NumRepArea*. The respective semantic conditions are specified in Figure H.1.

The first layers of the SES were described in Section 5.1. The SES was explained up to the abstraction hierarchy of the altitude of the watershed. Depending on the level of detail the *Altitude* can be of the type *SimpleAltitude* or of the type *Mountain1*, according to the two altitude abstraction hierarchy levels. Which specialization of the node *Altitude* is selected during pruning is defined by specrules in *AltitudeSPEC* evaluating the SESvar *MntLevel*.

Mountain1 describing the rainfall or snow depending on the height is composed of the four siblings: *Estimator, Basin, Areas*, and *UAM2*. The composition structure is specified with the aspect node *Mountain1DEC*. The leaf node *Estimator* refers to a basic model calculating the rainfall in each point of the mesh. The node *Areas* is refined by a variable number of nodes of type *Area*, each of which represents the rainfall on a specific considered area and refers to a basic model in the MB. Hence, *Areas* is followed by a multi-aspect node *AreasMASP*. In *AreasMASP* the variable number of children is specified with the SESvar *NumRepArea* and the coupling relations are defined in the attribute *cplg4*. *UAM2* is another leaf node referring to a basic model necessary for defining an abstraction hierarchy analogously to *UAM*. On this abstraction hierarchy level a time granularity is specified by different expressions of the inner entity *Basin*. It takes into account the storage effect of water or snow on the mountain. Thus, there is a specialization *BasinSPEC* in which dependent of the SESvar *BasinLevel* a selection is taken during pruning.

The resolution of the storage effect of the *Mountain* can be on a daily or hourly basis. If the time granularity shall be coarser, the leaf entity *SimpleBasin* is chosen. In case the watershed shall be studied with a finer time resolution, the entity *Basin1* is selected. *Basin1* is decomposed into the leaf entity node *SnowBasin*, which has the siblings *DAM3* and *UAM3* providing the same functionality as *DAM* and *UAM*.

Coupling relations between the entity nodes have the prefix *cplg* in Figure H.1 at aspect and multi-aspect nodes. They are presented in Figure H.2. For simplicity the types of the ports are left away, but the type of the coupling is specified as *External Input Coupling* (EIC), *Inner Coupling* (IC), or *External Output Coupling* (EOC). While the attributes *cplg1, cplg2* and *cplg5* are specifying static relations, *cplg3* and *cplg4* are affected by the varying number of replicated *Area* entities, that is specified by the multi-aspect *AreasMASP*. The current number of replications depends on the value of the SESvar *NumRepArea*. This SESvar is used to specify the varying coupling relations. The attributes *cplg3* and *cplg4* specify an SESfcn call and pass the SESvar *NumRepArea* as current input argument. As a representative for coupling functions *cplg3* specified by *cplfcn1* is discussed here in detail. In *cplfcn1*the SESvar *NumRepArea* is passed to the input variable *NUM*. The other two input variables are defined implicitly and encode the names of the parent node and

the subnodes of the node calling this function. In the *cplfcn* fixed and variable couplings depending on *NUM* are set.

A complete MB for the SES in Figure H.1 is presented in Figure H.3. However, no reference to DEVSimPy is made in the figure. The basic models with the prefix *Downward Atomic Model* (DAM) and *Upward Atomic Model* (UAM) are displayed with a bold frame. As can be seen in the SES the basic model *Est* needs to have a configurable number of output ports and the basic model *UAMmnt1* needs a configurable number of input ports. The number depends on the value in the SESvar *NumRepArea*.

Different PES specifying different levels of detail can be derived. For the highest level of abstraction with least detail the SESvar *WSLevel* is set to *0*. The *SimpleLayer* is selected and the watershed is not detailed further. Since the PES only has one layer, the PES equals the FPES. The resulting PES and its coupling relations are presented in Figure H.4 (a). Figure H.4 (b) shows a model generated from this PES.

The PES for the lowest level of abstraction with the most details is depicted in Figure H.5. The SESvars are set as depicted and the recalculated couplings are shown. During pruning the SESfcns are evaluated and static couplings are set in the respective aspect nodes. In the first abstraction *WS1* is selected, in the second abstraction *Mountain1*, in the third abstraction *Basin1*, and two *Area* nodes are generated. As discussed, deriving an FPES before applying the build method simplifies the resulting model. The flattening is thus advantageous for model generation. However, a model can also be generated from a PES. As in the PES, coupled components are part of the model accordingly. A model generated from the PES in Figure H.5 is depicted in Figure H.6. For a better overview the names of the ports are left away. Coupled components have no influence on model execution. Therefore the model generated from the PES is not minimal. A derived FPES for the lowest level of abstraction and its couplings is depicted in Chapter 5.1 in Figure 5.4 and the generated model is shown in Figure 5.5.

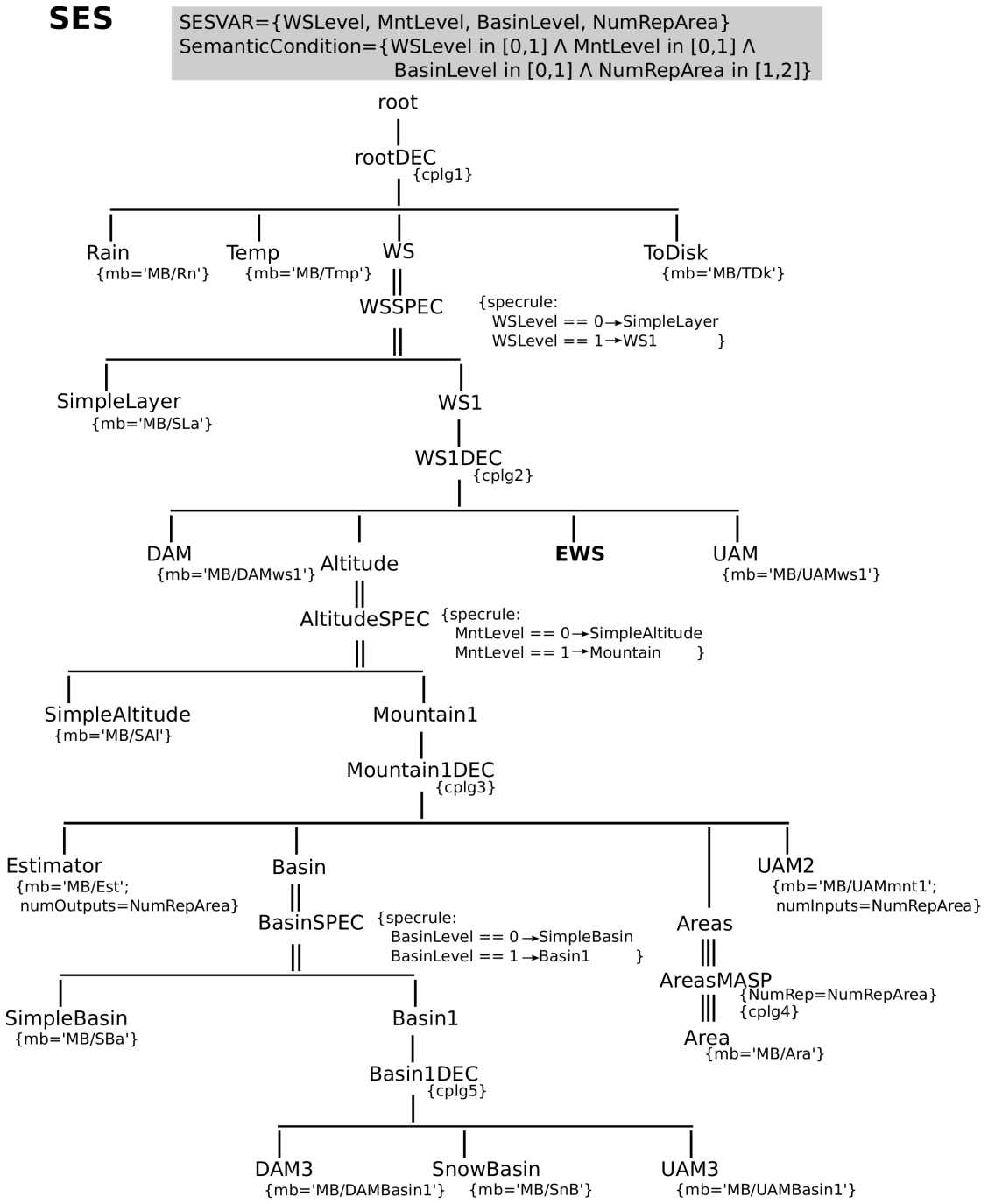


Figure H.1: SES for specifying model variants of the watershed with model components of different levels of detail.

```
cplg1={(Rain, out, WS, in1),
       (Temp, out, WS, in2),
       (WS, out, ToDisk, in)}

cplg2={(WS1, in1, DAM, in1),          # EIC
       (WS1, in2, DAM, in2),          # EIC
       (DAM, out1, Altitude, in1),    # IC
       (DAM, out2, Altitude, in2),    # IC
       (DAM, out3, Altitude, in3),    # IC
       (DAM, out4, Altitude, in4),    # IC
       (Altitude, out, EWS, in1),     # IC
       (EWS, out1, UAM, in1),         # IC
       (EWS, out2, UAM, in2),         # IC
       (EWS, out3, UAM, in3),         # IC
       (UAM, out, WS1, out)           # EOC }

cplg5={(Basin1, in1, DAM3, in1),          # EIC
       (Basin1, in2, DAM3, in2),          # EIC
       (DAM3, out1, SnowBasin, in1), # IC
       (DAM3, out2, SnowBasin, in2), # IC
       (SnowBasin, out, UAM3, in),    # IC
       (UAM3, out, Basin1, out1)      # EOC }
```

Couplings that are set via coupling functions:

variable NUMREP refers to the number of replications of the node calling the function -> refers to SESvar NumRepArea

```
cplg3=cplfcn1(CHILDREN,PARENT,NumRepArea)

cplfcn1(children,parent,NUM)

    # parent is Mountain1,
    # children(1) is Estimator,
    # children(2) is Areas,
    # children(3) is Basin,
    # children(4) is UAM2.

    # fixed couplings
    cplg(1)=(parent,in1,children(1),in1)          # EIC
    cplg(2)=(parent,in2,children(1),in2)          # EIC
    cplg(3)=(parent,in3,children(1),in3)          # EIC
    cplg(4)=(parent,in4,children(3),in1)          # EIC
    cplg(5)=(children(1),out1,children(2),in1)    # IC
    cplg(6)=(children(2),out1,children(3),in2)    # IC
    cplg(7)=(children(3),out1,children(4),in1)    # IC
    cplg(8)=(children(4),out,parent,out)          # EOC

    # variable couplings
    if NUM==2
      cplg(9)=(children(1),out2,children(2),in2)   # IC
      cplg(10)=(children(2),out2,children(4),in2) # IC
    end

return(cplg)
```

```
cplg4=cplfcn2(CHILDREN,PARENT,NUMREP)

cplfcn2(children,parent,NUM)

    # parent is Areas,
    # children(1) is Area_1,
    # children(2) is Area_2.

    k=1
    for i in range (1,NUM)
      cplg(k)=(parent,in(i),children(i),in)          # EIC
      cplg(k+1)=(children(i),out,parent,out(i))   # EOC
      k=k+2
    end

return(cplg)
```

Figure H.2: Coupling relations of the SES in Figure H.1.

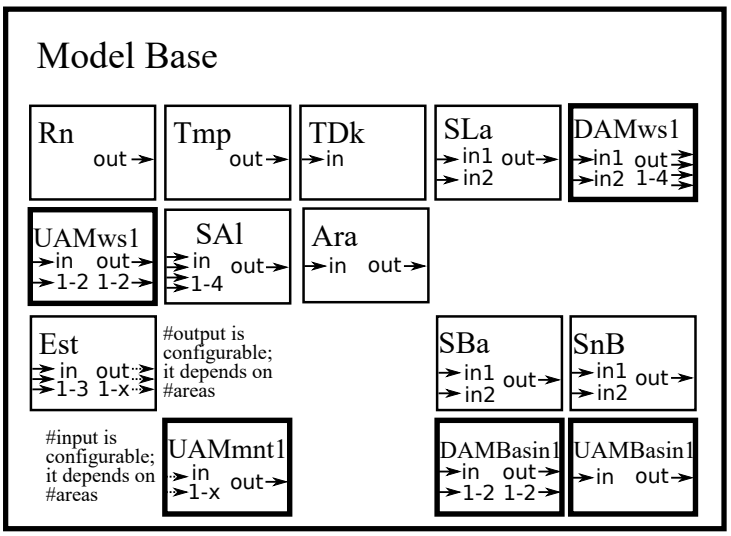


Figure H.3: MB for the SES in Figure H.1.

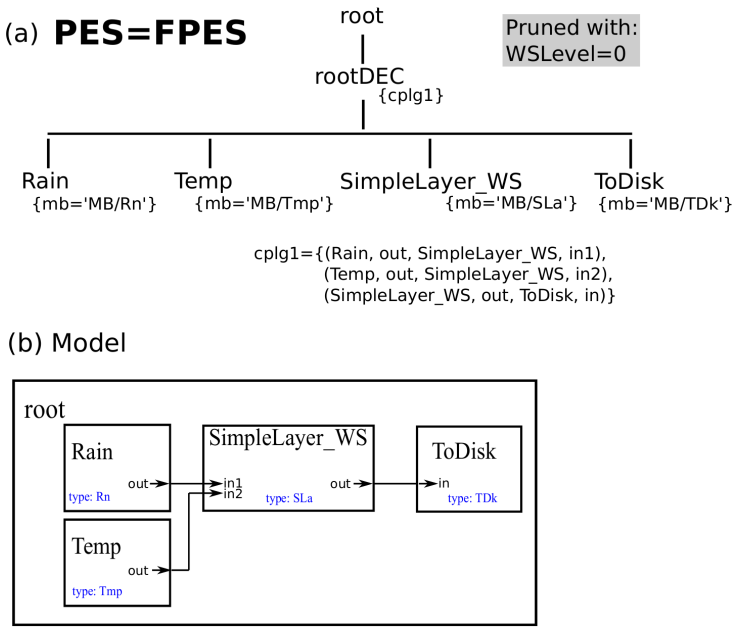
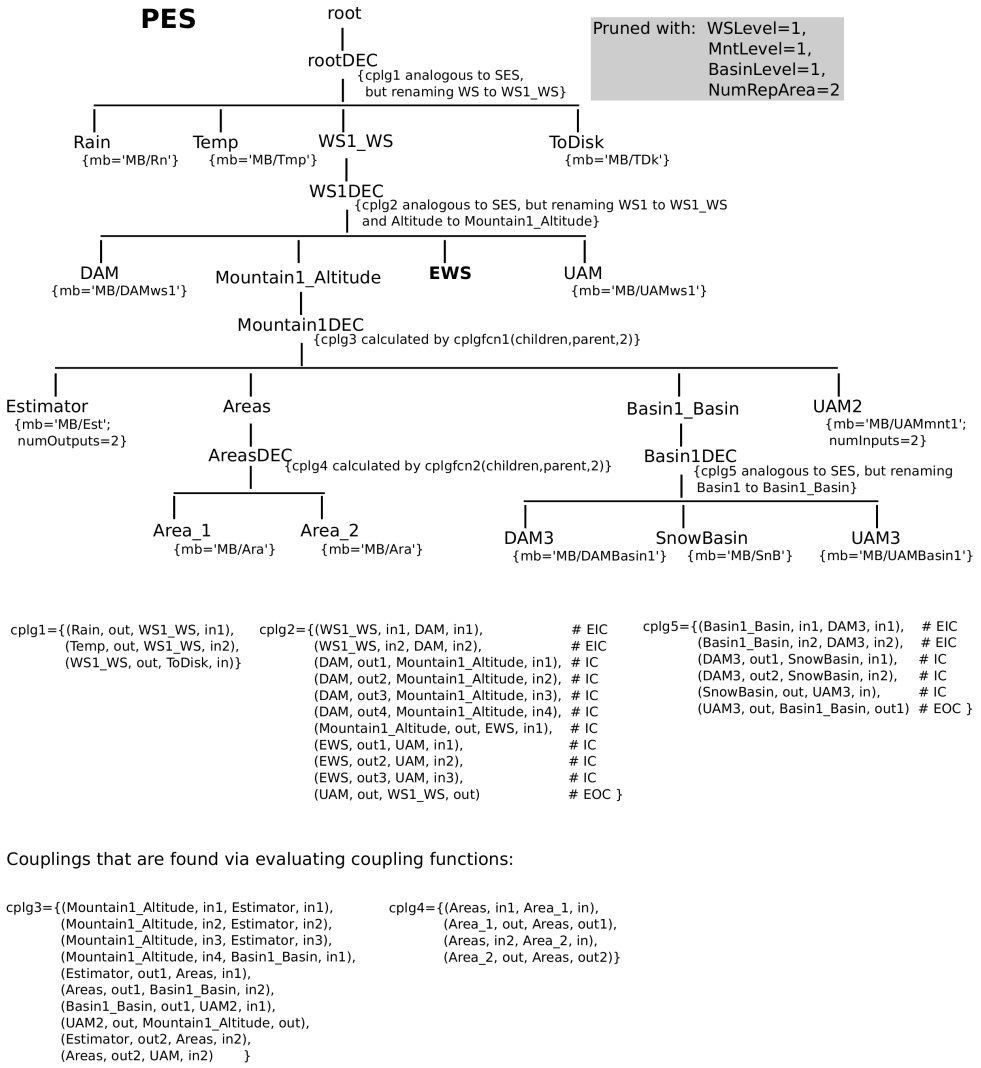


Figure H.4: PES and model for the highest abstraction level.




```
cplg1={(Rain, out, WS1_WS, in1),
       (Temp, out, WS1_WS, in2),
       (WS1_WS, out, ToDisk, in)}

cplg2={(WS1_WS, in1, DAM, in1),              # EIC
       (WS1_WS, in2, DAM, in2),              # EIC
       (DAM, out1, Mountain1_Altitude, in1), # IC
       (DAM, out2, Mountain1_Altitude, in2), # IC
       (DAM, out3, Mountain1_Altitude, in3), # IC
       (DAM, out4, Mountain1_Altitude, in4), # IC
       (Mountain1_Altitude, out, EWS, in1),  # IC
       (EWS, out1, UAM, in1),                # IC
       (EWS, out2, UAM, in2),                # IC
       (EWS, out3, UAM, in3),                # IC
       (UAM, out, WS1_WS, out)               # EOC }

cplg5={(Basin1_Basin, in1, DAM3, in1),   # EIC
       (Basin1_Basin, in2, DAM3, in2),   # EIC
       (DAM3, out1, SnowBasin, in1),     # IC
       (DAM3, out2, SnowBasin, in2),     # IC
       (SnowBasin, out, UAM3, in),       # IC
       (UAM3, out, Basin1_Basin, out1)   # EOC }
```

Couplings that are found via evaluating coupling functions:

```
cplg3={(Mountain1_Altitude, in1, Estimator, in1),
       (Mountain1_Altitude, in2, Estimator, in2),
       (Mountain1_Altitude, in3, Estimator, in3),
       (Mountain1_Altitude, in4, Basin1_Basin, in1),
       (Estimator, out1, Areas, in1),
       (Areas, out1, Basin1_Basin, in2),
       (Basin1_Basin, out1, UAM2, in1),
       (UAM2, out, Mountain1_Altitude, out),
       (Estimator, out2, Areas, in2),
       (Areas, out2, UAM, in2)       }

cplg4={(Areas, in1, Area_1, in),
       (Area_1, out, Areas, out1),
       (Areas, in2, Area_2, in),
       (Area_2, out, Areas, out2)}
```

Figure H.5: PES for the lowest abstraction level.

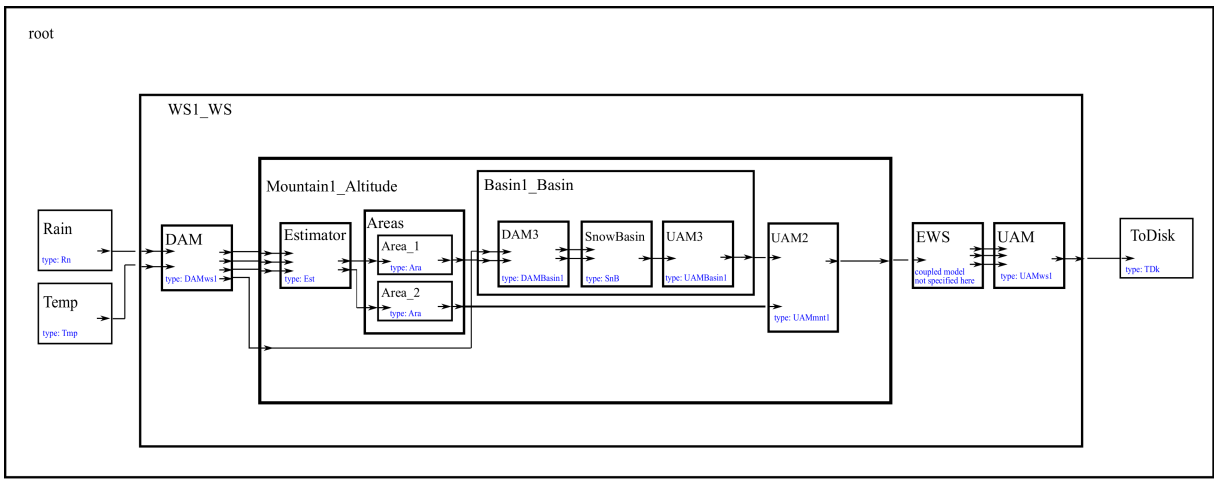


Figure H.6: Model derived from the PES for the lowest abstraction level.

I Software Generation with the SES/MB Framework

The description of non-simulation-specific component-based systems with SES/MB was introduced in Section 4.4.7 with focus on the generation of software applications. This is illustrated by the generation of a website. The website shows different variants of a *Hypertext Markup Language* (HTML) / JavaScript clock. The example is inspired by a *Weather Station*, which is discussed in the documentation of the software pure::variants. This software is mentioned in connection with feature modeling in Section 2.5.2.

Modeling of System Configurations The variants of the software structure are coded in the SES according to Figure I.1. The root node *clock* is composed of several components as specified by the node *clockDEC*. The clockwork in node *time.js* is mandatory. At this node the attribute *timezone* is specified by the SESvar *TZ*. The date specified by node *date.js* is optional. The selection depends on the specrule in node *dateSPEC*, which carries the selection out on basis of the value of the SESvar *Date*. The clock needs to have a display specified by node *index.html*, which represents the main function and entry point of the website. In addition a favicon is referenced in node *favicon.png*. A dark or a light style for the display can be set depending on the selection of the nodes *styled.js* or *stylel.js* in the specrule of node *styleSPEC*. The selection is based on the value of the SESvar *Style*.

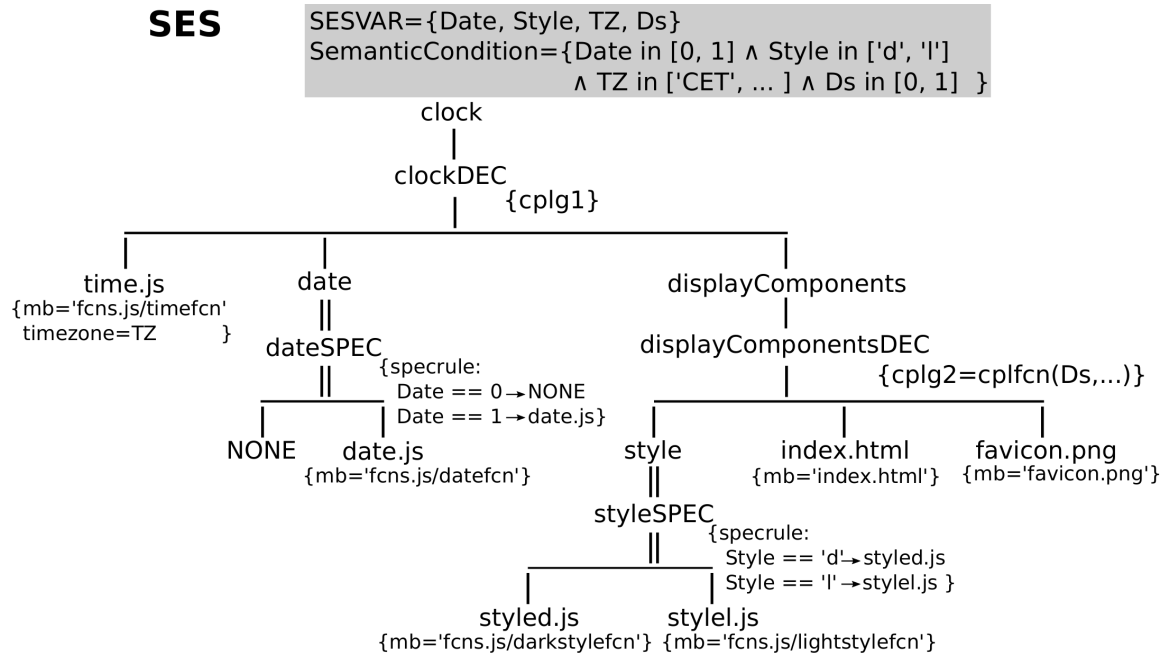


Figure I.1: An SES specifying software variants of a HTML / JavaScript clock.

The couplings *cplg1* and *cplg2* in nodes *clockDEC* and *displayComponentsDEC* need to be specified based on SESfcns. The coupling functions for the SES are not given here, because the way they are defined does not differ from the feedback control system example. In *cplg2* an excerpt of the function call is shown to illustrate the dependence of the coupling on the SESvar *Ds*, which selects the style of the date. However, the coupling relations are depicted as list for the FPES to discuss port types.

In an SES specifying a software application leaf nodes have the name of functions in the target language. These components are specified with functions from the MB according to the mb-attributes. For example the component *time.js* represents the JavaScript function *timefcn*, which is defined in *fcns.js* in the MB. Unlike model generation, the content of components is modified by inserting sourcecode. Other attributes specify values for variables in the respective functions. In the JavaScript function *timefcn* in the component *time.js* the variable *timezone* gets the value of the SESvar *TZ*.

Organizing an MB The MB organizes three files: the *index.html*, the *favicon.png*, and *fcns.js*. While the *index.html* organizes the main entry point and the *favicon.png* is a design element, the *fcns.js* organizes all JavaScript functions needed for the website.

Derivation of a Possible PES and Generation of the Appropriate Software Application Figure I.2 and Figure I.3 show a possible PES and the corresponding FPES. Couplings specifiy the structure of the software and ports represent input and output parameters of functions. Further information can be specified as port type. The couplings in *cplg* of the FPES are listed in Table I.1. The *darkstylefcn* in *styled.js_style* is only called once represented by the port type for a single call *sc*. The *timefcn* in *time.js* and *datefcn* in *date.js_date* functions are called regularely in an interval of *500 ms*. This is represented by the port type *ivl500*. The function *datefcn* is called with the input parameter *ds=0* in the *index.html* as specified in the SESvar *Ds*.

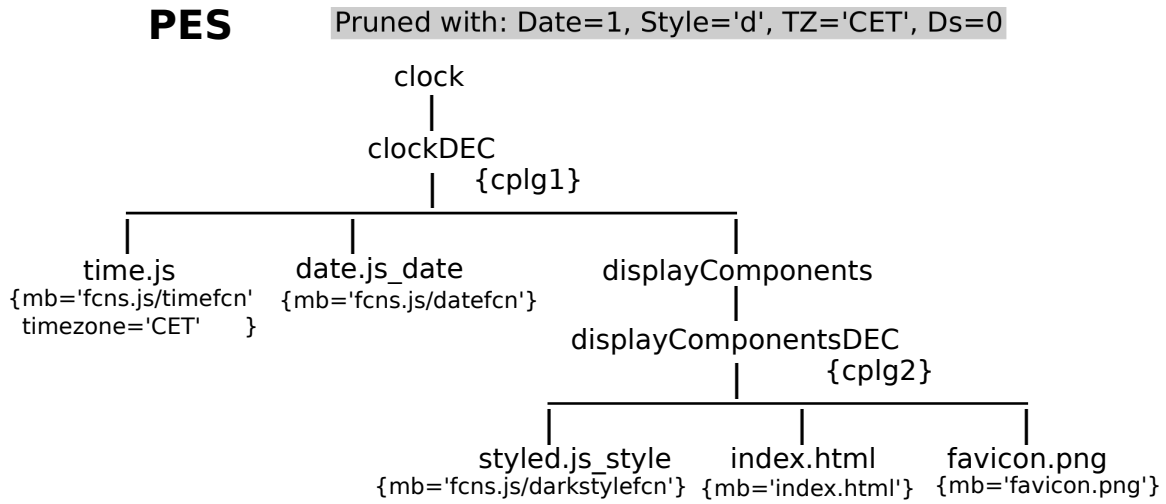


Figure I.2: A possible PES of a HTML / JavaScript clock.

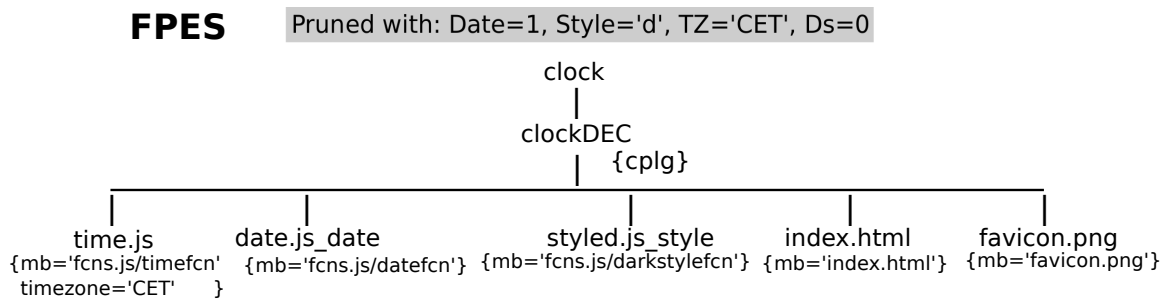


Figure I.3: A possible FPES of a HTML / JavaScript clock.

Table I.1: The couplings of the FPES in Figure I.3.

Source			*Sink*		
EntityName	*Port*	*Type*	*EntityName*	*Port*	*Type*
cplg					
styled.js_style	s	sc	index.html	s	sc
time.js	t	ivl500	index.html	t	ivl500
date.js_date	d	ivl500	index.html	d	ivl500
index.html	ds=0	ivl500	date.js_date	ds	ivl500

Due to the evaluation of the specialization nodes, the names of the nodes *date.js* and *styled.js* are extended. In the specific *model builder* for a target language, these extended names are shortened so that a component gets the name of a function in the target language. The names of the nodes *date.js_date* and *styled.js_style* are cropped to *date.js* and *styled.js*. The coupling relationships are adjusted analogously. The resulting software structure is shown in Figure I.4.

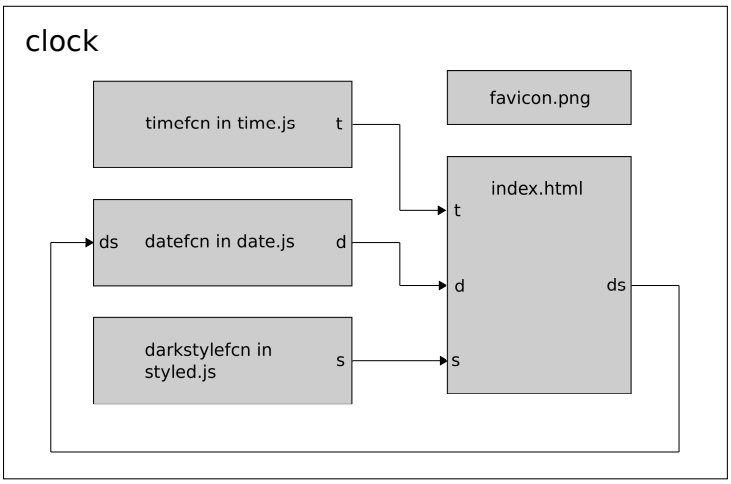


Figure I.4: Structure of the HTML / JavaScript clock according to the FPES in Figure I.3 and the introduced MB.

As shown in Figure I.4 the software is composed of several files. Their source code is shown next, but the picture *favicon.png* is not displayed. The entry point *index.html* is presented in Listing I.1 and the JavaScript functions in *styled.js*, *time.js*, and *date.js* are shown in Listings I.2, I.3, and I.4. This executable software application is generated by a native, target language specific model builder.

In lines 24–25 in the *index.hml* the *darkstylefcn* is called once and its return value is assigned to the variable *s*. The style is set in an *Cascading Style Sheets* (CSS) code wrapped as string in a JavaScript function. The function has the return value *s* as presented in Listing I.2. In lines 29–32 in the *index.hml* the *timefcn* is called in an interval of *500 ms* and its return value is assigned to the variable *t*. The function has the return value *t* as presented in Listing I.3. The parameter *timezone* is set to "*CET*" in the model builder according to an attribute in the FPES. In lines 36–39 in the *index.hml* the *datefcn* is called in an interval of *500 ms* and its return value is assigned to the variable *d*. It is called with the parameter value *ds=0*. The function has the input parameter *ds* and the return value *d* as presented in Listing I.4. Depending on the value of *ds* the style of the date can be

switched. A screenshot of the resulting website is presented in Figure I.5.

```
<!DOCTYPE html>
<html>
  <head>
    <!-- set page head -->
    <meta charset="UTF-8">
    <title>Clock</title>
    <link rel="icon" type="image/png" href="favicon.png" sizes="32
        x32">
  </head>

  <body>

    <!-- heading -->
    <p style="font-size:30px;color:red"><b>Clock</b></p>
    <p style="font-size:20px;color:red">Software Generation
        Example</p>

    <!-- set the outputs -->
    <output id="time"></output><br>
    <output id="date"></output>

    <!-- scripts depending on variant, code inserted by model
        builder -->

    <script language="javascript" type="text/javascript" src="
        styled.js"></script>
    <script>
      var s = darkstylefcn();
      document.head.appendChild(s);
    </script>
    <script language="javascript" type="text/javascript" src="
        time.js"></script>
    <script>
      setInterval(function() {
        var t = timefcn();
        document.getElementById("time").innerHTML = t;
      }, 500);
    </script>
    <script language="javascript" type="text/javascript" src="
        date.js"></script>
    <script>
      setInterval(function() {
        var d = datefcn(ds=0);
        document.getElementById("date").innerHTML = d;
      }, 500);
    </script>

  </body>
</html>
```

Listing I.1: The *index.html* HTML file as entry point for the website.

```
//output parameter is 's'
function darkstylefcn() {
    var s = document.createElement('style');
    //CSS style as string
    s.innerHTML = 'body {background-color: #000000;} output {color: #FFFFFF;}';
    //--------------------
    return s;
}
```

Listing I.2: The JavaScript *darkstylefcn* in the *styled.js* file.

```
//output parameter is 't', variable 'timezone' set
// by an attribute in the FPES
function timefcn() {
    function setZero(i) {
        if (i < 10) {
            i = "0" + i;
        }
        return i;
    }
    var today = new Date();
    var hour = today.getHours();
    var minute = today.getMinutes();
    var second = today.getSeconds();
    hour = setZero(hour);
    minute = setZero(minute);
    second = setZero(second);
    //---------------
    timezone = "CET";
    //---------------
    var t = timezone+" - "+hour+":"+minute+":"+second;
    return t;
}
```

Listing I.3: The JavaScript *timefcn* in the *time.js* file.

```
//input parameter is 'ds', output parameter is 'd'
function datefcn(ds) {
    function setZero(i) {
        if (i < 10) {
            i = "0" + i;
        }
        return i;
    }
    var today = new Date();
    var months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', '
        August', 'September', 'October', 'November', 'December'];
    var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', '
        Saturday'];
    var weekday = days[today.getDay()];
    var year = today.getFullYear();
    var month = months[today.getMonth()];
    var day = today.getDate();
    day = setZero(day);
    if (ds === 0) {
        var d = weekday+", "+year+"-"+month+"-"+day;
    } else {
        var d = weekday+", "+day+". of "+month+" in "+year;
    }
    return d;
}
```

Listing I.4: The JavaScript *datefcn* in the *date.js* file.

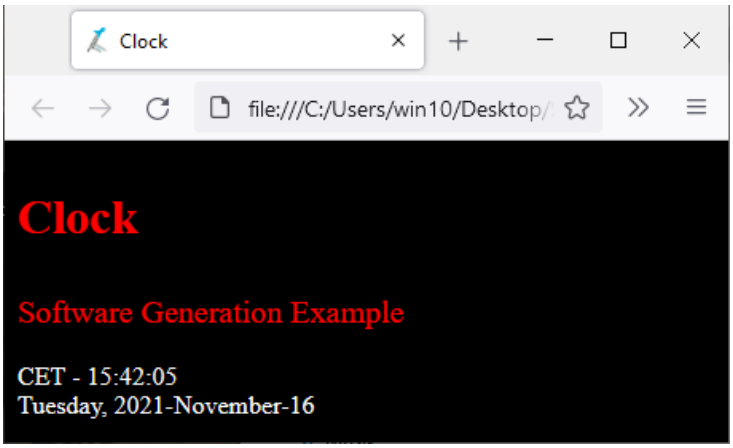


Figure I.5: A screenshot of the resulting website.

Statement of Authorship

Hiermit erkläre ich, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum Unterschrift