Model Generation for Multiple Simulators Using SES/MB and FMI

Hendrik Folkerts*, Thorsten Pawletta, Christina Deatcu

Research Group Computational Engineering and Automation, University of Applied Sciences Wismar, Philipp-Müller-Straße 14, 23966 Wismar, Germany; *hendrik.folkerts@cea-wismar.de*

Abstract. This paper deals with the extension of a Python-based infrastructure for studying the characteristics and behavior of families of systems. The infrastructure allows automatic execution of simulation experiments with varying system structures as well as with varying parameter sets in different simulators. Special focus is put on the support of different simulation environments by creating models implementing the Functional Mockup Interface (FMI). Possible system structures and parameterizations are defined using a System Entity Structure (SES). The SES as a high level approach for variability modeling, particularly in simulation engineering, describes a set of system configurations, i.e. different system structures and parameter settings of system components. In combination with a Model Base (MB), executable models can be generated from an SES. Based on the extended SES/MB approach, tool-supported variability modeling and automatic model generation and execution in different simulation environments using FMI is described. This is done by means of an engineering application.

Introduction

This paper is based on [1] and [2]. It focuses on the more general approach using the tools for variability modeling introduced there by integrating with the Functional Mock-up Interface (FMI) instead of just offering simulator specific solutions.

The high variant diversity with components of different application fields in today's technical systems leads to the need for variability modeling and integration of varying simulation platforms. One application area for variability modeling is e.g. the generation of software for electronic control units, which is often generated by underlying models. Those models are usually of similar type, but still differ in structure and parameterization. To handle modeling and simulation of these so called families of systems, several approaches for variabilty modeling exist. Most approaches make use of 150% models, which means that all possible behavior is put into just one large and complex model and functionality is then adjusted by switching off unneeded model parts. In contrast to 150% modeling, in this paper we describe a method to define, generate and simulate well-tailored and therefore lean models by making use of the System Entity Structure / Model Base (SES/MB) approach. The SES/MB approach [3] originates in the systems theory community and has undergone many extensions over the years [4, 5]. It allows platform-independent variability modeling with subsequent platform-dependent model generation of specific variants. The structures of systems are coded in an SES, while the dynamic models are organized in an MB. The SES links to these dynamic models.

For this approach, a proposal for using one MB in several simulators is detailed. This is achieved by creating an MB of models which implement the FMI. The general tool independent standard for *model exchange* and *co-simulation* FMI [6, 7] enables the exchange of models between different simulators. This makes it possible to combine models from different domains and execute them in several simulation environments.

After the extended SES/MB approach is briefly introduced, the paper presents some software tools implementing the theory. An engineering application example is then discussed in detail to clarify the process of model definition and model generation using FMI.

1 SES/MB Theory and Implementation

This section briefly discusses the general SES/MB theory and the derived extended SES/MB (eSES/MB) in-

ARGESIM Report 59 (ISBN 978-3-901608-93-3), p 13-20, DOI: 10.11128/arep.59.a59003

frastructure. Subsequently, an implementation of the infrastructure is presented.

1.1 SES/MB Basics and the eSES/MB Infrastructure

An SES is represented by a tree structure comprising entity nodes, descriptive nodes and attributes. A number of different system structures can be coded in one SES tree. In the context of modeling and simulation *entity nodes* are linked to *basic models* organized in an MB. Attributes of an entity node correspond to the parameters of the associated basic model. *Descriptive nodes* describe the relations among at least two entities and are divided into *aspect*, *multi-aspect* and *specialization* nodes.

In order to derive a specific system configuration all variation points are resolved by evaluating the rules at the descriptive nodes of the SES. This procedure is called *pruning*.

The resulting *Pruned Entity Structure (PES)* represents exactly one system configuration. In conjunction with an MB, a fully configured and executable model can be generated from the PES.

The basic SES/MB framework introduced in [3] was extended by new modeling features, methods and components [4, 5], such as an *Experiment Control (EC)* and an *Execution Unit (EU)* as shown in Figure 1. In this eSES/MB infrastructure, the EC uses an interface to the SES and its methods to derive goal-driven system configurations and to generate models, which are executed by the EU. The results returned by the EU are collected and analyzed by the EC. Thus, the derivation and generation of subsequent system configurations can be controlled reactively based on experiments already carried out.

A set of variables with global scope establish the interface to the SES. They are called *SES variables* (SESvar). *Semantic conditions* can be used to specify permitted value ranges and dependencies between SESvars. *SES functions* (SESfcn) are introduced for the specification of procedural knowledge. Complex variability can often be described more easily with SESfcns. Typical examples include the definition of varying coupling relations or the definition of variable parameter configurations in attributes. For automatic pruning, *selection rules* at descriptive nodes need to be defined, such as *aspectrules* for aspect and multi-aspect siblings or *specrules* at specialization nodes. A special mandatory attribute of multi-aspects is the attribute *number of*

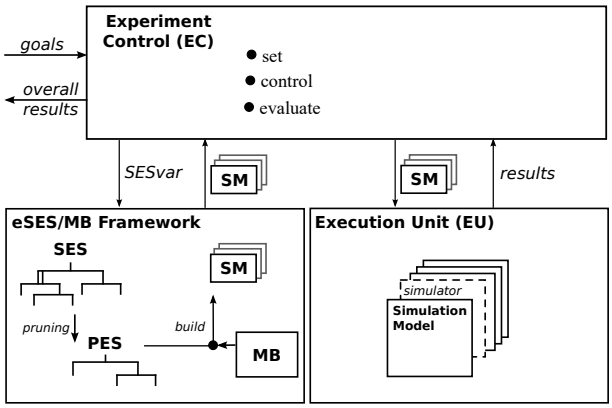

Figure 1: The eSES/MB infrastructure.

replications (numRep). The numRep attribute specifies the number of entities to create at a multi-aspect node during pruning. The *mb-attribute* of leaf entity nodes connects the entity node to a basic model in the MB. Attribute values and selection rules can be specified using SESvars or SESfcns.

1.2 Software Tools

The eSES/MB framework as presented in the lower left part of Figure 1 was implemented in a prototype software tool in MATLAB [8]. The focus of this tool is the modeling and generation of MATLAB/Simulink models. In contrast to the MATLAB prototype, the objective of the software used in this paper is to support the generation and execution of models for different simulation environments. The infrastructure in Figure 1 is implemented as a Python framework as presented in Figure 2. The tools are called *SESToPy*, *SESMoPy*, and *SESEuPy* [9].

SESToPy (**S**ystem **E**ntity **S**tructure **To**ols **Py**thon) implements a graphical editor and all SES related methods. In the editor an SES tree can be specified interactively in a file browser view and attributes and rules can be defined for every node. In addition to the pruning method already mentioned, SESToPy supports some more methods such as *merging* different SES and *flattening* for removing the hierarchy information. Applying the flattening method, a *Flattened Pruned Entity Structure (FPES)* is derived.

For generating executable models, SESMoPy (**S**ystem **E**ntity **S**tructure **Mo**del builder **Py**thon) was developed. SESMoPy is a model builder, which implements the *build* method in two different ways and

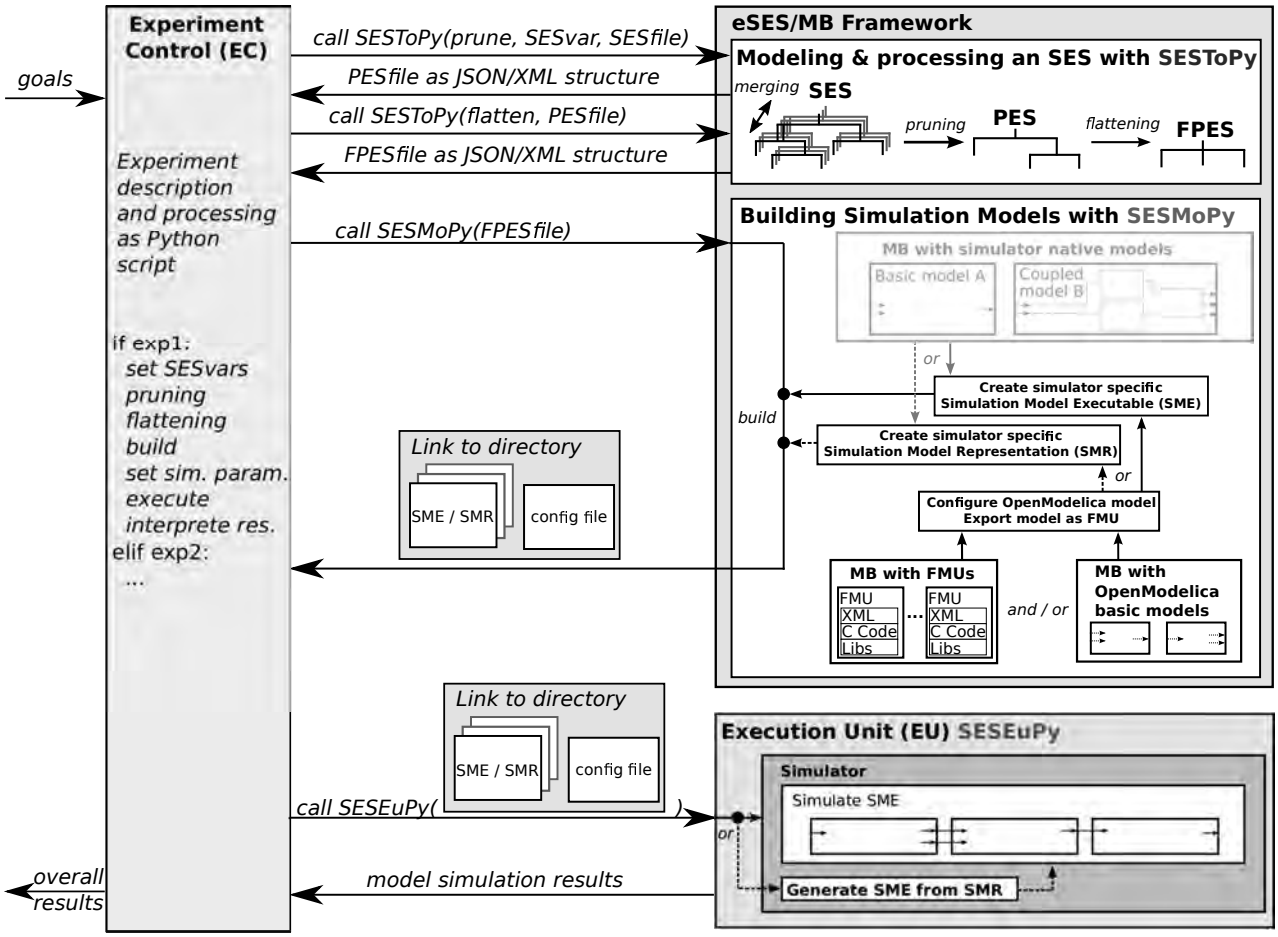

Figure 2: Python-based eSES/MB infrastructure for multiple EUs.

supports several simulation environments. For both approaches, all corresponding basic models must be organized in MBs, as shown in Figure 2. The first approach, called *native model generation*, is the generation of executables for a specific EU using a simulator specific MB. The second approach, this paper focuses at, is the *model generation based on FMI*. A Functional Mock-up Unit (FMU) is a model that implements an FMI [10]. In an FMU models are described by differential, algebraic, and discrete equations with time, state, and step events. In the scope of SESMoPy *FMI for Model Exchange* is used, which enables the simulation environment to generate C code of the FMU. *FMI for Co-simulation* is not discussed in this paper. The generalized interface FMI is supported by a number of established simulators [11], such as Simulink, OpenModelica or Dymola discussed for the use with SESMoPy.
Using the FMI-based approach, an MB with basic models from the simulator OpenModelica and/or an MB with FMUs are defined. SESMoPy creates an OpenModelica model and configures it according to the information passed in the FPES. Thus FMUs in an MB need to be imported into OpenModelica. The configured OpenModelica model is exported as FMU. Depending on the target simulator a specific Simulation Model Executable (SME) or Simulation Model Representation (SMR) is created. Finally SESMoPy returns a link to a directory, where the SME or SMR is placed together with a configuration file with information on the SME or SMR.
Information about the way the model is created can be provided in the EC calling SESMoPy or at the SES level according to the SES enhancements in [4].

The Python software tool SESEuPy (**S**ystem **E**ntity **S**tructure **E**xecution **u**nit **Py**thon) acts as a general EU. It implements a kind of wrapper for the integration of

different simulation environments into the framework. SESEuPy takes the link to the directory with the SME or SMR and reads the configuration file. If the model is given as SMR, an SME needs to be built. The SME then can be simulated in the target simulator and simulation results are returned.

In the next section, the components and functionality of the Python framework are explained using the example of an engineering application.

2 Engineering Application

A feedback control system can be modeled using transfer functions describing 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. The system can be mapped to a signal-flow oriented model. In the following paragraphs it is described how the eSES/MB infrastructure can be used to design and test such a system using the introduced tools and FMI-based model generation in combination with the simulation programs Matlab/Simulink, OpenModelica, and Dymola.

2.1 Problem Description

A process unit with a *PT1* behavior shall be controlled using a PID controller. A disturbance with a *PT1* behavior affects the output of the process unit. Different configurations of the PID controller shall be tested. If a defined regulatory goal is met, the current configuration of the PID controller is taken. Otherwise the structure is varied by adding a feedforward control to the system and different configurations of the PID controller are analyzed again. Figure 3 depicts a schematic representation of the application.

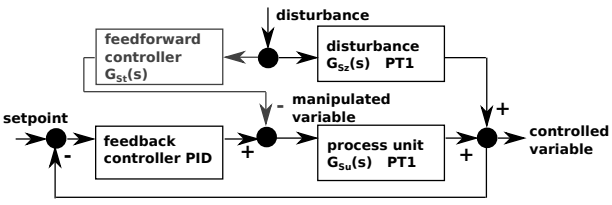

Figure 3: Structure of the feedback control system with optional feedforward control.

The system's behavior follows the *PT1* transfer function in Equation 1 and the step-shaped disturbance affects the output of the process unit with a *PT1* behavior according to Equation 2. The optional feedforward control is realized by subtracting the disturbing signal calculated by Equation 3 from the manipulated variable. The control goals are a settling time of less than 15 seconds and a maximum overshoot of less than 5% after a disturbance.

The system has two structure variants, either without or with the feedforward control part, and a range of different configurations for the PID controller can be applied for each structure variant. In the next section, the two structure variants and their possible configurations are specified as an SES.

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

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1} \tag{2}$$

$$G_{St}(s) = \frac{G_{Sz}(s)}{G_{Su}(s)} = \frac{20 \cdot s + 1}{10 \cdot s + 1} \tag{3}$$

2.2 Variant Modeling with SESToPy

The specification of the SES describing the feedback control system is done with the tool SESToPy. The tree and all attributes are defined via a graphical user interface. During modeling the SES with SESToPy, checks on the SES and plausibility tests are executed indicating model errors. The SES is saved as a JSON structure.

Figure 4 depicts the SES and its representation in SESToPy. The SES uses some extensions introduced in [5]. In addition to the different system configurations, essential parts for the configuration of simulation experiments are defined.

The root node *exp* of the SES and its subsequent aspect node *expDEC* describe a set of simulation based parameter studies for different system structures. The subtree of the entity node *simModel-ctrlSys* specifies the two system structures, i.e. a variant with and a variant without feedforward controller. The other two entity nodes specify experiment related information: The entity node *simMethod* specifies a target simulation environment for performing simulation runs using the SESvar *mysim*. The SESvar *myinterface* specifies whether to use the native or the FMI model generation. Other simulation execution parameters, such as the simulation period, are not specified and are set by the EC later. The entity node *expMethod* specifies the permitted value

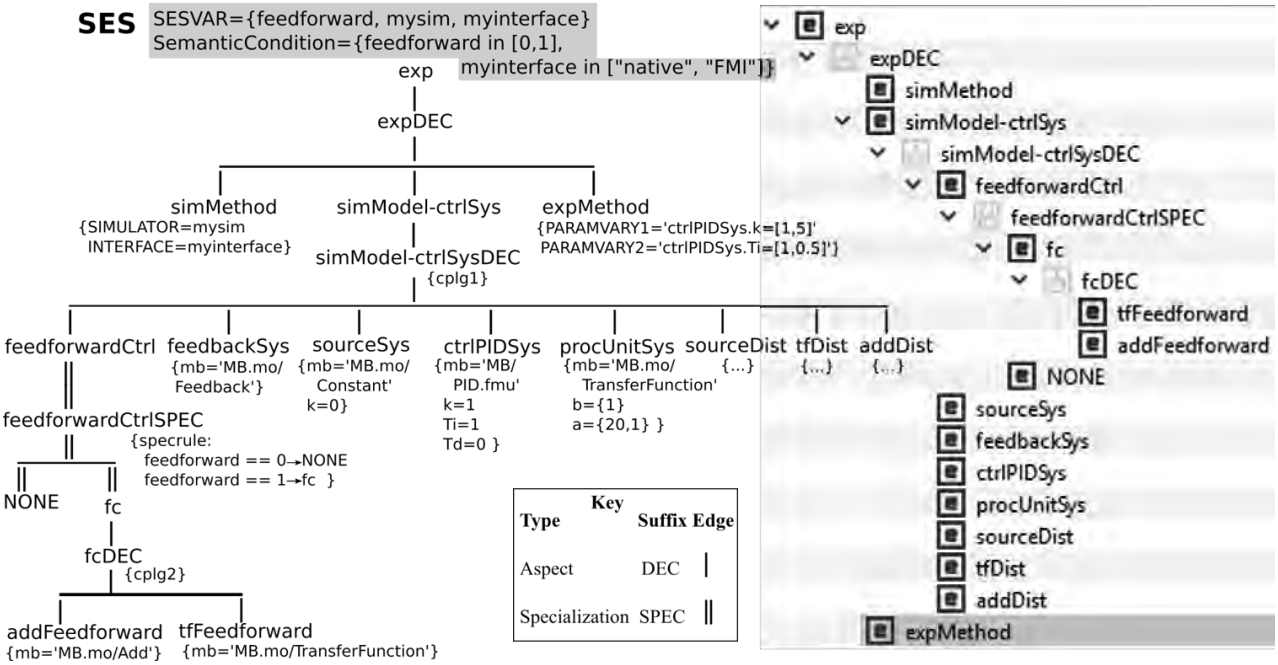

Figure 4: Left: SES specifying the feedback control system study; Right: Part of the SES representation in SESToPy.

ranges of two parameters for the PID controller. Besides the different system structures, they are the subject under study. The aspect *simModel-ctrlSysDEC* describes that each system variant consists of the following entities: *feedbackSys*, *sourceSys*, *ctrlPIDSys*, *procUnitSys*, *sourceDist*, *tfDist* and *addDist*. They are mandatory system elements. The optional feedforward control is specified by the subtree of entity *feedforwardCtrl*. The coupling relations of both structure variants are defined in the attribute *cplg1* of aspect *simModel-ctrlSysDEC*.

According to [12], optional parts in an SES are expressed by a specialization node where one of its children is a NONE element. A NONE element means that the entity is not included at all. The selection at a specialization is defined by an attribute called *specrule*. The specrule of the specialization *feedforwardCtrlSpec* defines that either the entity *fc* or *NONE* is selected during pruning. The result of evaluating the specrule at node *feedforwardCtrlSPEC* depends on the value of the SESvar *feedforward*. The SESvar codes the two possible structure variants as values 1 or 0. Therefore, the semantic condition $feedforward \in [0,1]$ applies to the SESvar. The entity *fc* and its subsequent aspect *fcDEC* specifies the feedforward control structure as a composition of the two entities *tfFeedforward* and *addFeedforward*.

Aspects and multi-aspects can define coupling relations as attribute. Couplings specify a composition of entities, which can be linked to basic models. Coupling attributes are abbreviated with *cplg* in Figure 4. Due to the varying system structures specified in the SES, the couplings in attribute *cplg1* of aspect node *simModel-ctrlSysDEC* are defined using an SESfcn. The coupling definitions in *cplg2* at node *fcDEC* are invariable and can therefore be defined without using an SESfcn.

According to Section 1, each leaf node defines an mb-attribute referring to a basic model in the MB. The basic model can be an OpenModelica component or an FMU. The other attributes of the leaf nodes define properties to configure the linked basic models. The values for *k* and *Ti* specified at node *ctrlPIDSys* are only default values, which will be overwritten because they are parameters under study.

2.3 Creating an MB

OpenModelica is an open source simulation platform and defines a set of basic models. It is widely used in different fields of engineering. In this case study OpenModelica basic models as well as FMU basic models are used. The FMUs define the FMI and can thus be

exported from any simulation environment.
For the OpenModelica MB a *package* is created that contains basic models. This package is stored as the *file MB.mo* and is referred to as *local OpenModelica library* in this paper. Furthermore the FMUs with the *fileending *.fmu* are stored in a folder on the local filesystem, which is referred to as *local FMU library* in this paper.

The local OpenModelica library is filled with the following basic models whose names correspond to the names in the mb-attributes of the leaf nodes in the SES:

- *Constant* as the setpoint for the controlled variable
- *Feedback* for closing the feedback control loop
- *TransferFunction* for representing the process, the disturbance's behavior, and the feedforward
- *Add* for adding signals

In the local FMU library FMUs are placed like listed. The names correspond to the names in the mb-attributes of the leaf nodes in the SES.

- *Step.fmu* for stimulating the disturbance
- *PID.fmu* is the controller of the feedback control system

Each basic model can be configured according to the attributes of the leaf node which they are linked to in the SES. The local OpenModelica library as well as the local FMU library act as MB for the basic models.

2.4 Experiment Execution

For executing simulation based experiments the experiment process and its goals need to be defined in a Python script. This script implements the EC according to Figure 2. The Python framework provides some EC related template scripts. The goals of the experiment were discussed in Section 2.1. The experiment should start with the study of different PID controller configurations using the control system structure without feedforward controller. The simulation is executed with the simulators OpenModelica, Dymola, and Simulink. In case that the objectives are not achieved by just varying the parameters *k* and *Ti* of the PID controller, the study shall be carried out with the additional feedforward control structure and the simulation programs OpenModelica, Dymola, and Simulink. A snippet of the EC script with essential steps of the experiment process is given next.

```
...
SESfile = ...
if conditions_for_experiment:
  #prune, flatten, build, and execute
  SESvar = [mysim = <simulator>,
            myinterface = "FMI",
            feedforward = 0]
  PESfile = SESToPy("prune",SESvar,
                                SESfile)
  FPESfile = SESToPy("flatten",PESfile)
  smHandle = SESMoPy("build",FPESfile)
  sim_param = [solver=<solver>, ...]
  results = SESEuPy("simulate",smHandle)
...
elif conditions_for_experiment:
  #prune, flatten, build, and execute
  SESvar = [mysim = <simulator>,
            myinterface = "FMI",
            feedforward = 1]
  PESfile = ...
  ...
...
```

The EC starts the experiment by setting the SESvars *mysim*, *myinterface*, and $feedforward$. A target simulator is set for *mysim*. Next, the EC calls SESToPy's API method for pruning with the current SESvar values and a reference to the file defining the SES as JSON structure. The pruning process results in a PES coded as JSON structure. Afterwards, the EC calls SESToPy's API method for flattening the PES. The created FPES is similar to the FPES shown in Figure 5, which represents the more complex FPES for the later SESvar assignment $feedforward = 1$. A reference to the file containing the FPES as a JSON structure is returned to the EC. The EC then calls SESMoPy's API method for the build method and passes the FPES file handle. SESMoPy determines the target simulator from the attribute at the node *simMethod* and the value ranges of the PID controller parameters under study from the attribute at node *expMethod* in the FPES.

Based on the information in the FPES and the basic models from the MB, SESMoPy creates an OpenModelica model for each configuration of the simulation model of the control system. FMU basic models need to be imported into OpenModelica. The configured OpenModelica model is exported as FMU, which is called *model FMU* in this context. This model FMU is simulator independent, since it implements the FMI. It represents an SM. Thus only one MB needs to be de-

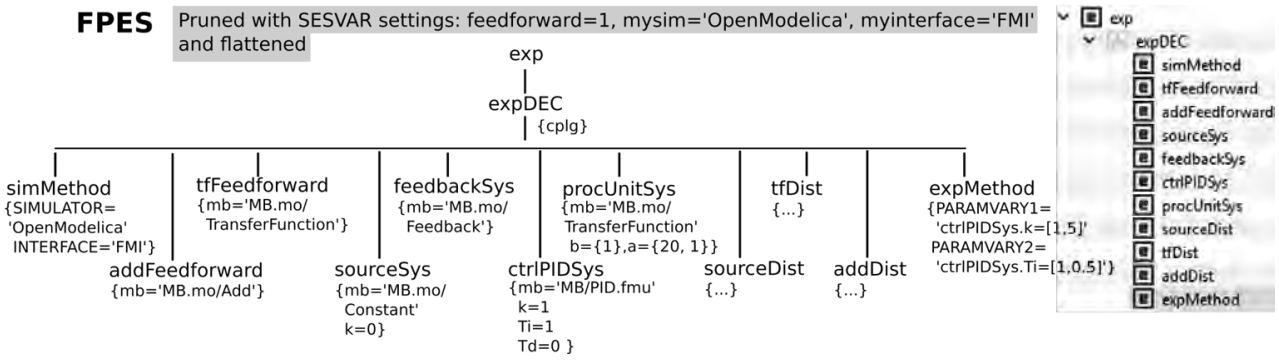

Figure 5: Left: FPES to study the feedback control system structure with feedforward; Right: FPES representation in SESToPy.

fined for use with multiple target simulators.

Depending on the target simulator different steps are necessary as discussed before. (i) A Simulation Model Executable (SME) for the target simulators OpenModelica and Dymola or (ii) a Simulation Model Representation (SMR) for the target simulator Simulink is created.
(i) The SME is built by importing the model FMU into the target simulator. Using the interface of the FMU simulator specific code is generated of the model. For execution a file with simulator specific instructions on the execution is generated. Furthermore a configuration file with information about the SME and its target simulator is created.
(ii) The SMR is a file with simulator specific instructions for the import of the FMU in the target simulator. The file is not executed yet. Furthermore a configuration file with information about the SMR and its target simulator is created.

SMs of one structure variant have different configurations of the PID controller. A handle to the directory with all SMs is returned by SESMoPy to the EC, referred to as *smHandle*. The EC extends the configuration file with simulation data, such as the solver to use or simulation start and stop time. The EC calls the tool SESEuPy and passes the smHandle as the link to the SMs and the configuration file. In collaboration with the target simulation environment, SESEuPy controls the execution of an SM. An SME can be executed directly, whereas during execution of an SMR an SME is built. Figure 6 shows the structure of a fully configured OpenModelica model, but with feedforward controller, i.e. for the SESvar assignment $feedforward = 1$. Finally, SESEuPy returns the simulation results to the EC.

In case the results meet the experimental goals, the overall results are calculated and returned by the EC. In case the goals are not reached, the second system structure with the additional feedforward controller by the SESvar assignment $feedforward = 1$ is set and a new model configuration and generation is started.

If the experimental goals have been achieved, the overall results of the experiment are the necessary control structure and the appropriate PID controller parameter settings. Otherwise the failure to achieve the objectives may also be established.

In addition simulation with another simulator can be tested. In the SESvar *mysim* another simulator is set and the model generation and simulation process is started over with the structure variant without feedforward controller. In this way, model by model validation is achieved using different simulators.

3 Conclusion

In this paper the extension for working with FMI of some Python-based software tools for variant modeling are presented. The entire process of variability modeling beginning with the system specification with an SES up to automatic variant derivation, model building, and execution is described. The proposed eSES/MB infrastructure makes it possible to model and simulate engineering problems using different target simulation environments.

References

[1] Folkerts H, Deatcu C, Pawletta T, Hartmann S. Python-based eSES/MB Framework: Model Specification and Automatic Model Generation for

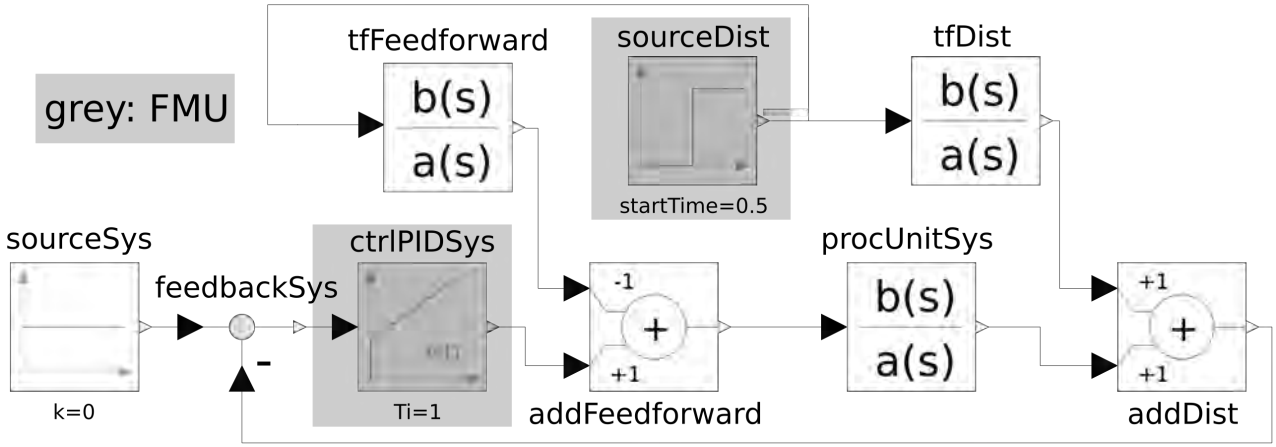

Figure 6: OpenModelica SM of the feedback control system with feedforward control.

Multiple Simulators. *SNE Simulation Notes Europe.* 2019; 29: 207–215. doi: 10.11128/sne.29.tn.10497.

[2] Folkerts H, Deatcu C, Pawletta T, Hartmann S. A Python Framework for Model Specification and Automatic Model Generation for Multiple Simulators. *2019 International Interdisciplinary PhD Workshop*; 2019 May; Wismar. IEEE. doi: 10.1109/IIPHDW.2019.8755423.

[3] Zeigler BP, Kim TG, Praehofer H. *Theory of Modeling and Simulation.* 2nd ed. Cambridge: Academic Pr.; 2000. 510 p.

[4] Schmidt A, Durak U, Pawletta T. Model-Based Testing Methodology Using System Entity Structures for MATLAB/Simulink Models. *SIMULATION: Transactions of The Society for Modeling and Simulation International.* 2016; 92(8): 729–746.

[5] Schmidt, A. *Variant Management in Modeling and Simulation Using the SES/MB Framework* [dissertation]. University of Rostock; 2019.

[6] Blochwitz T, Otter M, Arnold M, Bausch C, Clauß C, Elmqvist H, Junghanns A, Mauss J, Monteiro M, Neidhold T, Neumerkel D, Olsson H, Peetz JV, Wolf S. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In Proc. of the 8th International Modelica Conference. *Modelica Conference*; 2011 March; Dresden, Germany. p 105–114. doi: 10.3384/ecp11063.

[7] Blochwitz T, Otter M, Akesson J, Arnold M, Clauß C, Elmqvist H, Friedrich M, Junghanns A, Mauss J, Neumerkel D, Olsson H, Viel A. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In Proc. of the 9th International Modelica Conference. *Modelica Conference*; 2012 Sept; Munich, Germany. p 173–184. doi: 10.3384/ecp12076173.

[8] Pawletta T, Pascheka D, Schmidt A, Pawletta S. Ontology-Assisted System Modeling and Simulation within MATLAB/Simulink. *SNE Simulation Notes Europe.* 2014; 24: 59–68. doi: 10.11128/sne.24.tn.102241.

[9] Research Group CEA. *Python-Based eSES/MB Infrastructure.* 2019. www.github.com/hendrikfolkerts, last accessed 2020/09/10.

[10] Modelica Association Project "FMI". *Functional Mock-up Interface for Model Exchange and Co-Simulation.* 2014. https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf, last accessed 2019/08/06.

[11] Modelica Association Project "FMI". *Functional mock-up interface for model exchange and co-simulation.* 2019. https://fmi-standard.org/tools/, last accessed 2020/02/21.

[12] Deatcu C, Folkerts H, Pawletta T, Durak U. Design Patterns for Variability Modeling Using SES Ontology. In Proc. of Spring Simulation Multi-Conference 2018. *Spring Simulation Multi-Conference*; 2018 Apr; Baltimore/MD, USA. SCS Int. p 3:1–3:12.