Python-based eSES/MB Framework: Model Specification and Automatic Model Generation for Multiple Simulators

Hendrik Folkerts[1*], Thorsten Pawletta[1], Christina Deatcu[1], Sven Hartmann[2]

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

[2]Department of Informatics, Clausthal University of Technology, Julius-Albert-Straße 4, 38678 Clausthal-Zellerfeld, Germany

SNE 29(4), 2019, 207-215, DOI: 10.11128/sne.29.tn.10497
Received: July 20, 2019 (Selected EUROSIM 2019 Postconf. Publ.), Accepted: September 22, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. This paper proposes 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. Possible system structures and parameterizations are defined using a System Entity Structure (SES). The SES is a high level approach for variability modeling, particularly in simulation engineering. An SES 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 an extended SES/MB approach, an enhanced software framework is introduced that supports variability modeling and automatic model generation for different simulation environments. By means of an engineering application it is shown, how a set of Python-based open source software tools can be used to model an SES and to automatically generate and execute signal-flow oriented models.

Introduction

This paper is a modified version of [1]. It clarifies the scope of the discussed tools for variability modeling.

Generally, variability modeling can be seen as an approach to describe more than one system configuration. A system configuration incorporates the structure of the model as well as the parameter settings. Different system configurations arise e.g. when modeling varying real world systems or when modeling system variations for finding an optimal system design.

In software engineering, a classical approach to variability modeling is the use of Feature Models (FM) in combination with 150% models. Feature Models specify components and relations, which are used for modeling the variability of a system or product [2]. By selecting features a Variant Model (VM) can be generated from the FM. Thus, the VM represents a subset of all variants defined in the FM. The FM and the VM are platform-independent. For generating executable models, platform dependent dynamic models are needed, which are linked to the FM. All dynamic models are organized as an 150% model. With the help of a variant generator an executable model is generated based on a VM and the 150% model. The software pure::variants is a prominent example supporting this approach [3]. Another approach is the executable Unified Modeling Language with its standard Foundational Unified Modeling Language (fUML). The fUML is a subset of the graphical notation of the UML with executable semantics in order to generate platform dependent software [4]. Based on the UML the executable Systems Modeling Language (SysML) was developed in the field of systems engineering.

Analogous to approaches coming from software engineering the systems theory community introduced methods for platform-independent variability modeling with subsequent platform-dependent model generation of specific variants. One method is the System Entity Structure (SES) and Model Base (MB) approach. In this paper the theory of an extended SES/MB approach is discussed and software tools implementing the theory are presented using an engineering application example.

1 SES/MB Theory and Implementation

This section briefly discusses the general SES/MB theory and the derived extended SES/MB (eSES/MB) infrastructure. 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. Different system structures can be coded in an 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. An aspect node represents the composition of an entity. The multi-aspect node is a special aspect node and describes the decomposition of an entity into several entities of the same type. The specialization node describes the taxonomy of an entity. Descriptive nodes can specify variation points using specific rules.

Specifying a composition of entities, which describes the composition of models in a modular-hierarchical manner, requires a specification of *coupling* relations. Couplings can be specified as properties of descriptive nodes of the type aspect or multi-aspect and consist of pairs of entity names and port names, such as *(source entity, source port, sink entity, sink port)*.

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*. If more than one aspect or multi-aspect node appear on the same tree level – so called siblings –, they form a variation point with an xor-selection. A second kind of variation point for an xor-selection is defined by specialization nodes. When selecting one child entity of the specialization, the name and the properties of the selected child entity are merged with the name and properties of the father entity of the specialization. This procedure is formalized by the SES inheritance axiom. 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 [5] was extended in [6] and [7] by new modeling features, methods and components, 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.

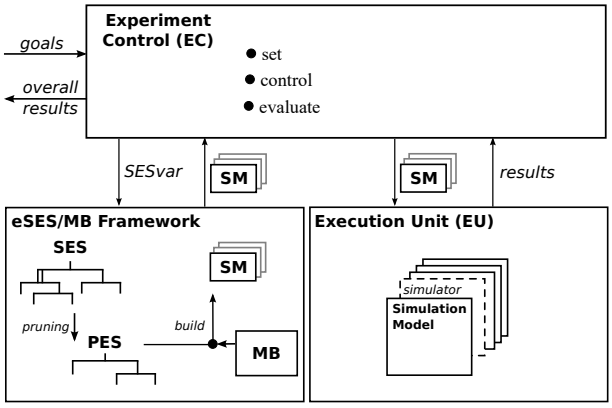

Figure 1: The eSES/MB infrastructure.

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 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.

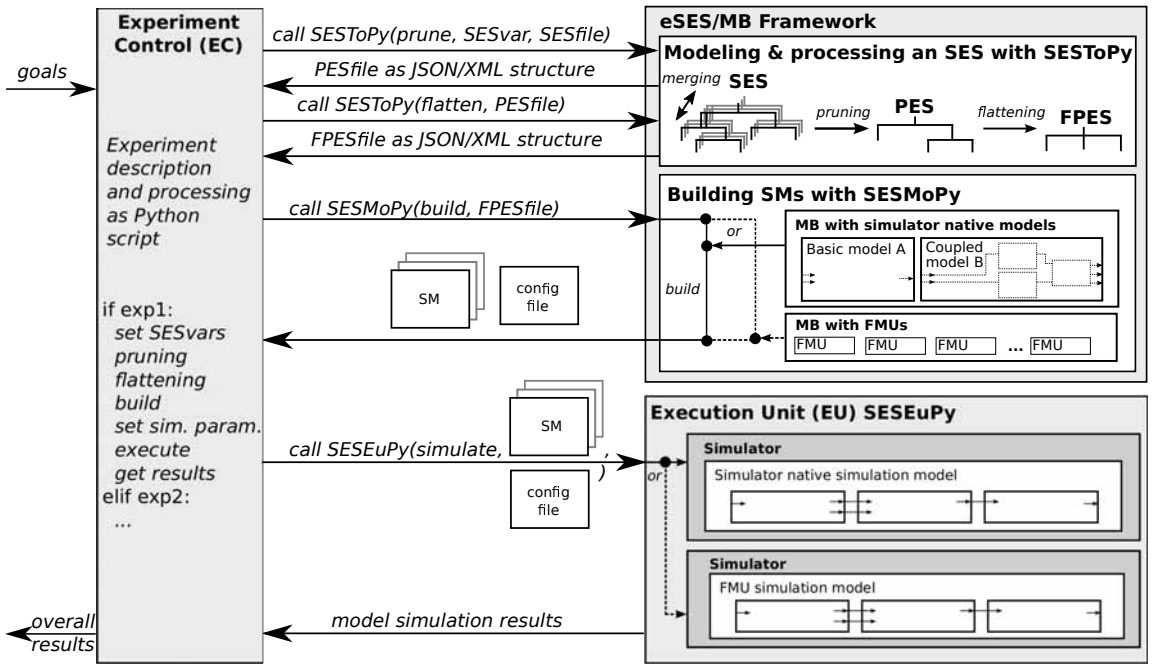

Figure 2: Python-based eSES/MB infrastructure for multiple EUs.

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 new prototype discussed 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. The pruning and flattening methods can be used interactively or as API methods. SESToPy supports to save SES, PES, and FPES as JSON or XML files.

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 supports several simulation environments. For both approaches, all corresponding basic models must be organized in an MB, as shown in Figure 2. The first approach, called *native model generation*, is the generation of executables for a specific EU. Currently native models for *MATLAB/Simulink, Dymola, OpenModelica*, the *PDEVS for MATLAB* toolbox and the Python-based environment *DEVSimPy* [10] can be generated. Currently for MATLAB/Simulink, Dymola, and OpenModelica SESMoPy only supports the build method for the generation of signal-flow oriented models. Support for the generation of physical models is under development. The second approach is the *model generation based on the Functional Mock-up Interface (FMI)*. The FMI standard is defined for several uses, one of which is FMI for Model Exchange. The generalized interface FMI is supported by a number of established simulators [11]. Using this approach, SESMoPy builds a Functional Mock-up Unit (FMU) model that implements an FMI [12]. This approach is under development. Depending on the way of model generation, the processing in the corresponding EU is different.

SESMoPy's build method generates executable simulation models (SM) using the information in the FPES and basic models from a target specific MB. 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 [7].

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. The modular structure and well-defined interfaces allow to integrate even components of non-simulation-specific environments into the framework.

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 native model generation for OpenModelica.

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.

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.

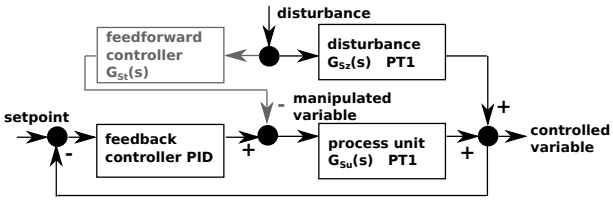

Figure 3: Structure of the feedback control system with optional feedforward control.

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 [13]. 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

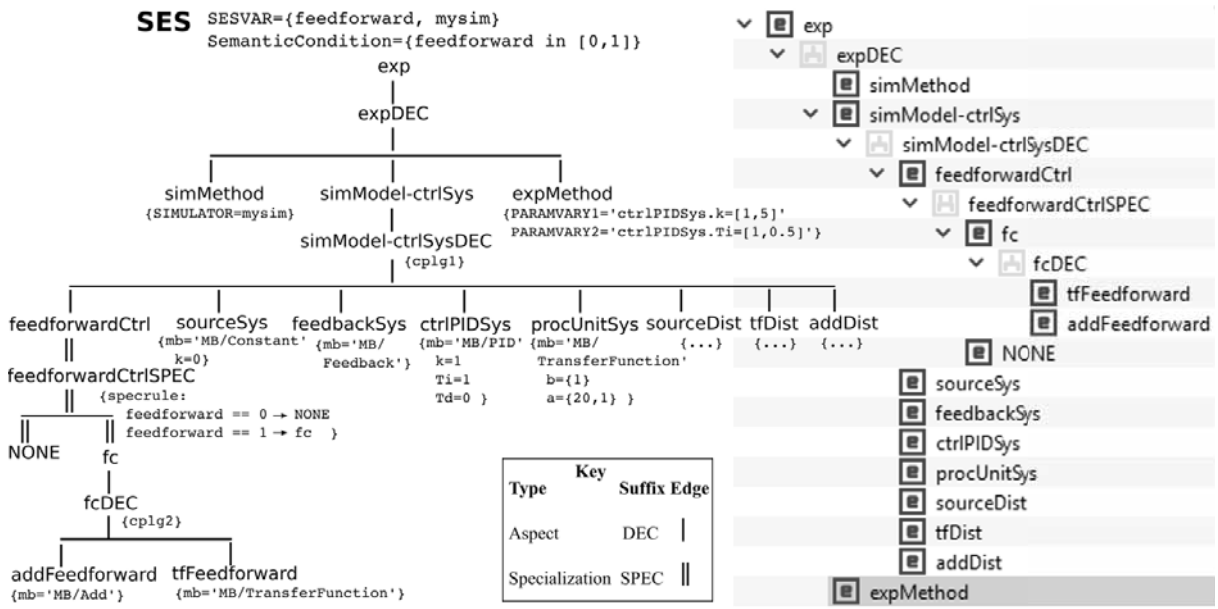

Figure 4: Left: SES specifying the feedback control system study; Right: Part of the SES representation in SESToPy.

environment for performing simulation runs using the SESvar *mysim*. 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 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: *sourceSys*, *feedbackSys*, *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*, as discussed later.

According to [14], 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. These 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 following code is an excerpt of the SESfcn.

```
def cplgfcn(feedforward):
  cplg = []

  #fixed couplings
  cplg.append(["sourceSys","y",
       "feedbackSys","u1",""])
  cplg.append([...

  #variable couplings
  if feedforward==0:
    cplg.append([...
  elif feedforward==1:
    cplg.append([...

  #return
  return cplg
```

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 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 Basic Models with OpenModelica

OpenModelica defines a set of basic models for different fields of engineering. An OpenModelica *package* is created that contains all basic models. Although this leads to a duplication of some components, this makes the model base independent of potential future changes in OpenModelica libraries. The package 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
- *Step* for stimulating the disturbance
- *Feedback* for closing the feedback control loop
- *PID* is the controller of the feedback control system
- *TransferFunction* for representing the process, the disturbance's behavior, and the feedforward
- *Add* for adding signals

Each basic model can be configured according to the attributes of the leaf node which they are linked to in the SES. The package acts as an MB for OpenModelica 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. 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. A snippet of the EC script with essential steps of the experiment process is given next.

```
...
SESfile = ...
if conditions_for_experiment:
  #prune, flatten, and build
  SESvar = {"mysim": "OpenModelica",
       "feedforward": 0}
  PESfile = SESToPy("prune",SESvar,
       SESfile)
  FPESfile = SESToPy("flatten",PESfile)
  smHandle = SESMoPy("build",FPESfile)
  #execute
  sim_param = ...
  results = SESEuPy("simulate",smHandle)
...
elif conditions_for_experiment:
  #prune, flatten, and build
  SESvar = {"mysim": "OpenModelica",
       "feedforward": 1}
  PESfile = SESToPy("prune",SESvar,
       SESfile)
  FPESfile = SESToPy("flatten",PESfile)
  smHandle = SESMoPy("build",FPESfile)
  #execute
  sim_param = ...
  results = SESEuPy("simulate",smHandle)
...
```

The EC starts the experiment by setting the SESvar *mysim* and $feedforward$. 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.

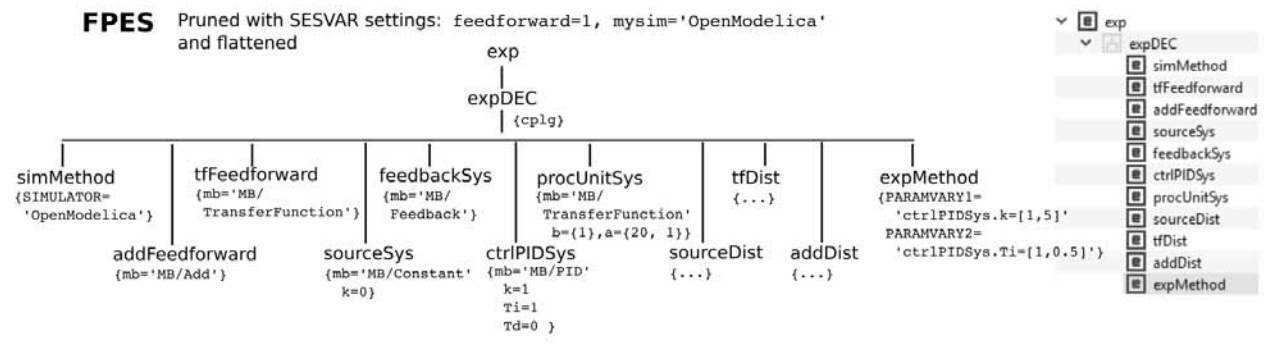

Figure 5: Left: FPES to study the feedback control system structure with feedforward; Right: FPES representation in SESToPy.

Based on the information in the FPES and the basic models from the MB, SESMoPy creates multiple files. For each configuration of the simulation model of the control system one target simulator specific file containing the model and its configuration is created. Simulation models of one structure variant have different configurations of the PID controller. Furthermore a configuration file containing information about the created models and the target simulator is created. A handle to all the created files 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, etc. Then, the EC calls the tool SESEuPy and passes the smHandle as the link to the model files and the configuration file. In collaboration with the target simulation environment OpenModelica, SESEuPy creates the fully configured simulation model and controls its execution. 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. Otherwise a new model configuration and generation needs to be started. For this purpose, the *elif* part in the code snippet defining the EC defines the experiment steps for the second system structure with the additional feedforward controller by the SESvar assignment $feedforward = 1$.

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.

Figure 7 depicts the simulation results of the feedback control system resulting from the execution of the presented EC. In parts (a) and (b) the simulation results of the system structure without feedforward control and with different PID controller parameter settings are presented. The required control goals of a settling time of less than 15 seconds and a maximum overshoot of less than 5% after a disturbance as specified in Section 2.1 for the tested PID controller parameter settings could not be achieved. In part (c) of Figure 7 the simulation results of the system structure with feedforward control is presented. The required control goals are reached and the PID controller parameter settings are returned as overall results.

3 Conclusion and Future Work

In this paper free software tools for variant modeling are presented, beginning with the system specification with an SES up to automatic variant derivation, model building and execution. The eSES/MB infrastructure has been implemented as prototype Python tools. They make it possible to model and simulate engineering problems using different target simulation environments. Future works will especially cover support for physical modeling, a deeper integration of FMI into SESMoPy and test other target simulation environments.

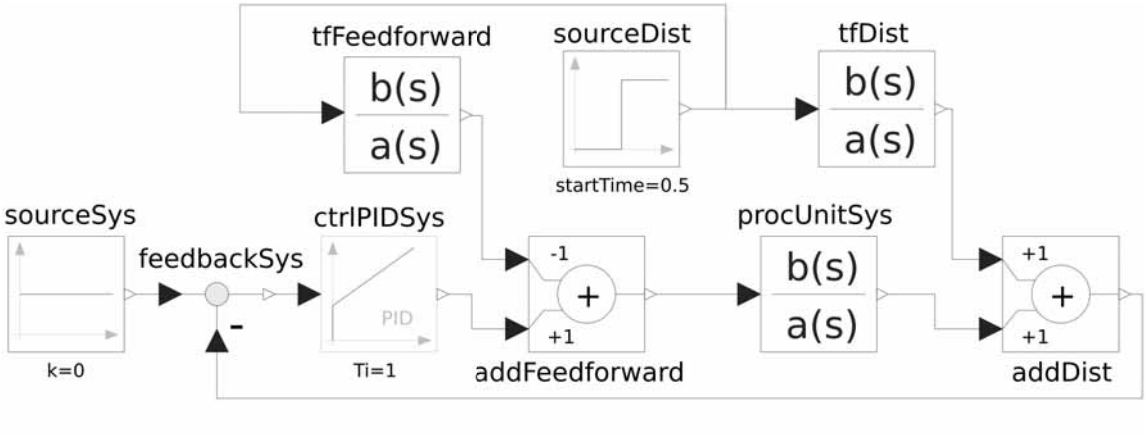

Figure 6: OpenModelica SM of the feedback control system with feedforward control.

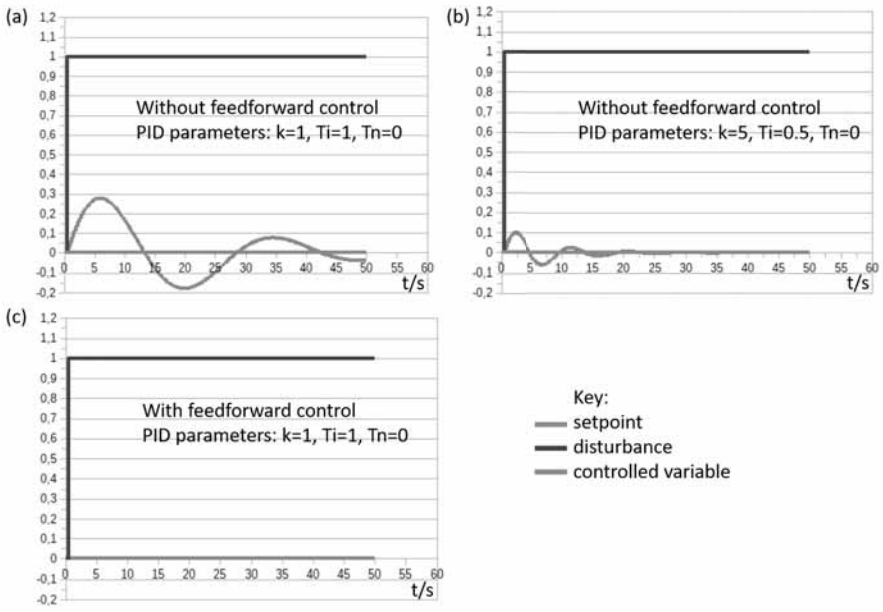

Figure 7: (a) and (b): Without feedforward control and with different PID controller settings, (c): With feedforward control.

Acknowledgement

The authors acknowledge the grant from the German Science Foundation, DFG (PA 631/2). Moreover, the authors would like to thank Peter Junglas, who contributed valuable work to the development of a generic model builder, Daniel Pascheka, who implemented a first version of the graphical editor for MATLAB, Birger Freymann, who redesigned the MATLAB editor and implemented a model builder for the MatlabDEVS toolbox, and our former colleague, Tobias Schwatinski, who provided preliminary work. Last, but not least, we would like to thank Bernie Zeigler for motivating and supporting our work.

References

[1] 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.

[2] Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, SE Inst. Carnegie Mellon Univ. Pittsburgh/PA, USA. 1990.161 p.

[3] pure-systems GmbH. *Technical White Paper: Variant Management with pure::variants.* 2006. https://www.pure-systems.com/mediapool/pv-whitepaper-en-04.pdf, last accessed 2019/08/06.

[4] Mellor SJ, Balcer M. *Executable UML: A Foundation for Model-Driven Architectures.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 2002.

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

[6] Pawletta T, Schmidt A, Zeigler BP, Durak U. Extended Variability Modeling Using System Entity Structure Ontology Within MATLAB/Simulink. In Proc. of Spring Simulation Multi-Conference 2016. *Spring Simulation Multi-Conference*; 2016 Apr; Pasadena/CA, USA. SCS Int. p 62-69.

[7] 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.

[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 Infrastruture.* 2019. http://www.cea-wismar.de/tbx/Py_eSESMB/, last accessed 2019/08/06.

[10] Capocchi L, Santucci JF, Poggi B, Nicolai C. DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems. *20th International WETICE Conference.* 2011 June; Paris. 170–175. doi: 10.1109/WETICE.2011.31.

[11] Modelica Association Project "FMI". *Functional mock-up interface for model exchange and co-simulation.* 2019. https://fmi-standard.org/tools/, last accessed 2019/08/06.

[12] 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.

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

[14] 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.