University of Applied Sciences Wismar
University of Applied Sciences
Technology, Business and Design
Research Group Computational Engineering and Automation

Documentation

SESMoPy - SES Modelbuilder Python

A Python Modelbuilder Using an SES and an MB

as of: April 28, 2019

by: Research Group CEA

Contents

1 Introduction **3**

2 Working with SESMoPy **6**
2.1 Creating the Model Base 6
2.2 Connection to the SES 7
2.3 Building 9
2.4 Building an Executable / Runnable of SESMoPy in Windows and Linux 13

3 Background: Concepts for modelbuilding from an SES for different simulators **14**

4 Examples **17**
4.1 Example #01: Simple Model - Aspect Node 17
4.2 Example #02: Feedback Control System with Optional Feedforward Control 18
4.3 Example #03: Feedback Control System with Optional Feedforward Control - FMI 20

5 Related Work **21**

Bibliography **22**

List of Figures **23**

List of Tables **24**

List of Listings **25**

Acronyms **26**

1 Introduction

Color in text: Quick'n dirty solution, will be changed. In development. To describe.

This is the modelbuilder SESMoPy, generating executable models for Flattened Pruned Entity Structure (FPES) generated with the SES modeling software SESToPy. For information on the System Entity Structure (SES) and its extensions to the extended SES/MB (eSES/MB) infrastructure please read the documentation of the SES modeling software SESToPy. This software was written in the Research Group Computational Engineering und Automation (CEA) at the University of Applied Sciences Wismar.
It is written in Python 3 using the bindings PyQt5 for the user interface. It is programmed in Python 3.4.1 with PyQt 5.5, but as of August 2018 it runs in current Python/PyQt versions as well. There are scripts for building an executable in Windows using cx_Freeze and a runnable in Linux using PyInstaller. Curently there are no more modules needed than given in the Python Standard Library.
As shown in detail in the documentation of SESToPy the SES can be connected to an Model Base (MB) for generating models. It was extended to allow automatic model generation and execution of models, depicted in Figure 1.1.

This software provides the *build* method, generating models with different parameterization from one structure variant derived as an FPES. This is shown in Figure 1.2 for clarification. The focus is on the support of different simulators.
A detailed explanation on the Python-based software structure supporting the eSES/MB infrastructure is given in the documentation of SESToPy.

One FPES representing one structure variant can encode several system configurations. For one structure variant, encoded in an FPES, for values of attributes of entity nodes, which refer to parameters of the basic models, a range of values can be defined (already in the SES). This leads to a number of simulation models (SM).

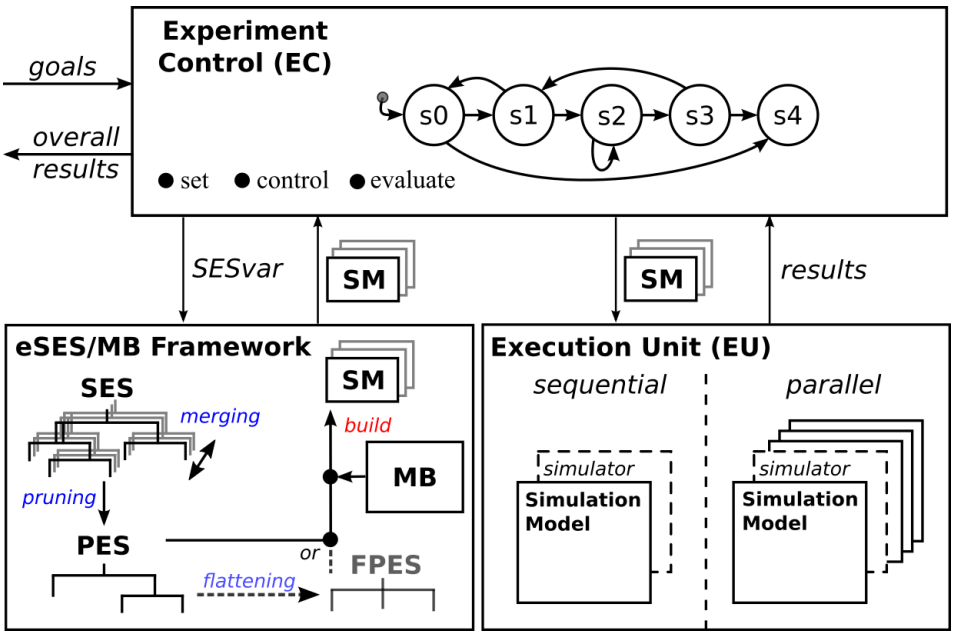

Figure 1.1: Extended SES/MB-based infrastructure.

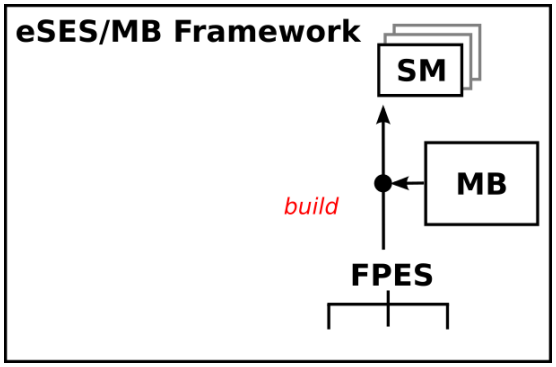

Figure 1.2: Part of the eSES/MB infrastructure supported by SESMoPy.

Currently, this software supports the creation of models for the simulators *Simulink*, *OpenModelica* and *Dymola*. Dymola can use the Modelica Model Library. SESMoPy further supports modelbuilding using the Functional Mock-up Interface (FMI) as a general interface. The FMI is supported by a number of simulators and is defined as FMI for Model Exchange and FMI for co-simulation. SESMoPy uses the FMI APIs of Simulink, OpenModelica and Dymola to build models using Functional Mock-up Unit (FMU)s for model exchange. FMUs are the basic models implementing the FMI. An FMU is a zipped file with the file extension “fmu“. An FMU contains an XML file defining the FMI as well as C-code and libraries (.dll / .so)).
In this documentation the creation of models without using FMI is called “native“ model creation, using FMI is called “FMI“ model creation.

MATLAB/Simulink has an API for Python with the name “MATLAB Engine API for Python“, OpenModelica has a Python interface called “OMPython“ with the PySimulator and Dymola has a Python interface as well. However, the goal is to use as less external libraries as possible, since they usually require a special Python version number. In order to avoid incompatibilities SESMoPy uses the native interfaces of the simulators.

2 Working with SESMoPy

In the following sections of this chapter information on how to use SESMoPy in combination with the different simulation programs are given.
As seen in Figure 1.2, for modelbuilding there is a need for creating an MB. The MB is specific for the simulator. Using the native interface for each simulator one MB needs to be created, whereas using FMI the same MB can be used without adaption with several simulators (since the simulators support FMUs defining the FMI). The connection to the SES is specified and the build method described.

2.1 Creating the Model Base

In this section there is a description for the supported simulators on how to create an MB in the respective programs. In these simulation programs, a basic model in the MB is called "block". In Section 2.2 is clarified how the simulator specific MBs need to be adjusted so that one SES can be used for multiple simulators in the native interface. When using FMI, only an OpenModelica modelbase needs to be created.

Simulink Open Simulink and create a blank model. Place the blocks needed for the SES in the model and save it. Creating subsystems of the blocks may be needed. See examples in Sections 4.1 and 4.2 for details. An "Out" block needs to be added, in order to view and return simulation results. Save it with the name, the MB shall get.

OpenModelica Open OpenModelica Connection Editor (OMEdit). On the left side in the Libraries Navigator make a right click on the blank space. Select "New Modelica Class". In the popup window give a name for the class (when saving, the file should get the same name later). Select "Package" as specialization. Fill the class with the blocks needed for the SES. Therefore: Right click the new created class and select "New Modelica Class". Enter a blockname, select "Block" as specialization,

and for “Extends (optional)“ find the block in the Modelica library this block shall be derived from. Make sure, it is derived from the right library block! Repeat, until all blocks are placed in the new library. Finally, save it with the same filename as the name of the library.
Make sure, that the file containing the just created MB has the same name as the created package.

Dymola Open Dymola. In the Dymola program select File -> New -> Package. In the popup window give a name for the package (when saving, the file should get the same name later). Fill the package with the blocks needed for the SES. Therefore: Right click the new created package and select New -> Block. Enter a blockname and for Extends find the block in the Modelica library this block shall be derived from. Make sure, it is derived from the right library block! Repeat, until all blocks are placed in the new library. Finally, save it with the same filename as the name of the library.
Make sure, that the file containing the just created MB has the same name as the created package.

2.2 Connection to the SES

The connection to the SES, which is derived as FPES, is made with the MB attribute as introduced in the documentation of SESToPy.
A leaf node gets an attribute with the name “mb“ and the value “modelbasename/blockname“. The value needs to be in quotation marks. The moldebasename is the same name as the filename of the file(s) containing the MB. A whole path with folders starting from the directory containing the FPES file can be set. The blockname is the name of the block defined in the MB (see above) Therefore the node does not need to be called like the block in the MB. The MB shall lie in the directory defined in the mb-attribute starting from the directory with the .jsonsestree file containing the FPES.

Further attributes configure the block. Parameters of a block can be set with attributes of the node, which refers to the block by the mb-attribute.

The connection between blocks is specified in the couplings of the FPES (derived from an SES) specifying the connection of ports of blocks.

In the documentation of SESToPy it is stated that the SES shall be independent of the simulator the model is built for. As written above, nodes referring to basic models (blocks) often need to define a value for a block parameter. Furthermore blocks are connected using the couplings of the SES. Couplings define which ports of blocks to connect.
For usage in different simulators *using the native interface*, one modelbase for each simulator is created. The basic models for different simulators need to have the same interface (same parameters, same ports) to be applicable for the same SES (which defines the configuration of basic models and their couplings). Therefore, the simulator specific modelbases need to be adjusted. Ports can be adapted by creating coupled models as one block in the MB so that the basic models for each simulator have the same portnames. The parameters can be adapted by a function defined for the MB and executed before simulation. See the files of the examples #01 and #02 in Sections 4.1 and 4.2 for details.
For usage in different simulators *using FMI*, an OpenModelica MB is created. The SES configures the basic models in this OpenModelica MB. The configured basic models are translated to FMUs using an OpenModelica API function. The directory containing a number of FMU files is called MB in this context. The preconfigured FMUs in this MB define the standardized interface FMI and are applicable for all supported simulators (not all simulators supporting FMI, since the simulator API for FMI is used). The FMUs have simulator independent ports. Couplings in the SES are defined with these ports. When the FMUs are imported in the target simulator, simulator specific syntax for further configuration is necessary. Further configuration would be done manually and is usually not needed. See example #03 in Section 4.3 for details.

It needs to be defined which simulator and interface to use.

Names of the block parameters The names of the block parameters in the libraries can be found like described next. For Simulink just look in the documentation. E.g. for a step block just search for “simulink step block“ and find the parameter name listed as “Block Parameter“. The type of the values are listed as well. The step time for example is called “Time“ and needs a character vector as input. For OpenModelica right click the block and select “Open Class“. All parameter names are listed there (can be seen even in sourcecode). The step time of the step block for example is called “startTime“ in OpenModelica. For Dymola it is the same as with OpenModelica for Modelica blocks.

However, in order to use generalized names for block parameters in the SES, the modelbases for different simulators need to be adjusted as written above. Using FMI the SES needs to define the parameters and syntax to configure the MB defined in OpenModelica. In the build process blocks are exported as preconfigured FMUs interpreteable for each simulator supporting FMI.

Portnames Every block has inputs and outputs. When building the model, the ports of blocks need to be connected. The portnames of the blocks can be found like the parameter names: In the documentation for Simulink, in the class of the block for OpenModelica and the same for Dymola for Modelica blocks. E.g. the step block has the port with the name 1 in Simulink and the port with the name y in OpenModelica and Dymola (since it is a Modelica block).
In order to use generalized portnames in the SES, the modelbases for different simulators need to be adjusted as written above. Using FMI the portnames are defined in the FMU implementing the FMI (derived from an OpenModelica block) and are therefore the same for every simulator supporting FMI.
Simulink has two lists of portnumbers, one for normal ports and one for entity ports. In the SES define portnames in the couplings in the form "N" for normal ports and "CN" for entity ports, where "N" represents an integer.

2.3 Building

SESMoPy has a very small interface allowing to select an FPES as .jsonsestree file, call this documentation and start the build process. A field for a status message is reserved. The interface is presented in Figure 2.1. An FPES file is selected and the

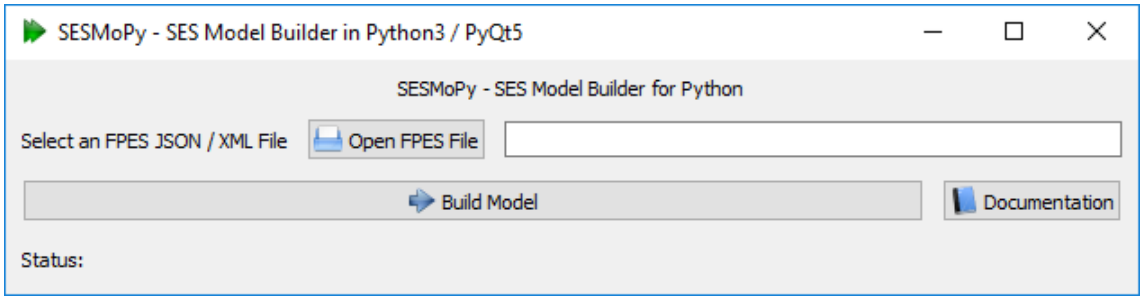

Figure 2.1: SES/MB supported by SESMoPy.

modelbuilding process is started with click on the button "Build Model". The file(s)

containing the MB need(s) to be placed in the path defined in the mb-attributes starting from the directory containing the FPES file as discussed in Section 2.2.

For modelbuilding in the SES a node defining the simulator in the attributes “SIMULATOR“ and “INTERFACE“ and a node defining varying parameters as attributes “PARAMVARY“ for one structure variant need to be defined. The parameters defined in “PARAMVARY“ are combined. Every combination leads to one SM. The other part of the SES needs to define nodes with an MB attribute at their leaves. See examples section! -> Use of Experimental Frame (EF) will change this

Use of EF to control model generation!

The SIMULATOR can be “Simulink“, “OpenModelica“, or “Dymola“. The INTERFACE can be “native“ or “FMI“.

In the build process for a structure variant (an FPES) a folder is created in which models of the same structure variant but different configuration are placed. The folder gets the name of the FPES file with an appended “_models“.
The FPES is analysed and textfiles with instructions defining the SM are created. The modelfile(s) get name(s) of the of the FPES file with an appended string _model_ and a number of the model. All models with different configurations, but of one structure variant (one FPES) are in one folder.

Processing the MB Furthermore the MB is copied in the modelfolder. Using the native interface the MB(s) is a file / are files organizing the simulator specific basic models as described before. They just need to be copied, since the configuration of the basic models they organize is done in the simulator specific SM. Using FMI each block from the OpenModelica MB is placed in one file, the blockconfiguration of the blocks (=basic models) is added, and finally the blocks are exported as (configured) FMUs and placed in a subdirectory of the modelfolder. The directory containing these FMUs is referred to as MB for the FMUs. When using OpenModelica or Dymola as simulator these FMUs are imported in the respective simulator and .mo files of the imported FMU are created.

Created modelfile in Simulink A Simulink model is created as Matlab script .m file which can start Simulink, add and configure blocks and connections. With the first commands in the file a new Simulink model is created.
A block is added with the command: add_block(’modelbase/blocktype’, ’modelname/blockname’); When the native interface is used, attributes configuring block

parameters are placed as variables in the script. These variables are named “blockname_parametername“. An additional function defined for the modelbase reads the variables and configures the block they are defined for.
Using FMI the imported FMU Simulink block is configured with the name of an FMU file. The folder containing the FMU file needs to be on the Matlab path. Each imported FMU is placed in a subsystem in order to get named ports. The name of the subsystem is adapted. The name of the ports of a subsystem corresponds to the name given in the couplings in the SES. Some code is necessary to find the correct portnumber from a given portname. The connections are drawn with the add_line('modelname', outportHandle, inportHandle) function.

Created modelfile in OpenModelica and Dymola A model for OpenModelica and Dymola is in a file with the fileending .mo and begins with the keyword *model* and the name of the model. After that the blocks are defined.
An entry follows the structure: modelbase.blocktype blockname(configuration);
Blocktype and blockname may not be identical. When using FMI as interface no block configuration is written. The definition of the blocks ends with the keyword *equation*. In the next part of the file the connections are defined.
An entry follows the structure: connect(blockname.sourceport, blockname.sinkport);
The model description ends with the keyword *end* and the name of the model.

Configuration file For every structure variant a configuration file with the information according to Table 2.1 is written in the created folder. The configuration file holds basic information on the SM. The first word of every line indicates the information given in the line.

Table 2.1: Entries in the configuration file.

Variable	Description
MODELNAMEPARAM	Modelname and the model's parameter variation
MODEL	Complete path to a model file (*.m or *.mo)
SIMULATOR	“Simulink“ or “OpenModelica“ or “Dymola“
INTERFACE	“native“ or “FMI“
MODELBASE	Complete path to a modelbase file (or folder containing FMUs)

There can be several entries “MODEL“, “MODELBASE“, and “MODELNAMEPARAM“, since there can be several models with varying parameterization, each of which uses

basic models from different modelbases. The “SIMULATOR“ entry specifies the simulator to use and the “INTERFACE“ entry specifies the interface to use.
In case the native INTERFACE is used, one MODELBASE entry specifies a file containing the modelbase.
In case the FMI INTERFACE is used, the modelbase entry needs to define the folder containing the preconfigured FMUs of each variation. Therefore the MODEL for which the MB is configured is written in brackets in the entry as well.
The configuration file can be extended.

Return value SESMoPy returns the filepath of the modelfolder containing modelfile, MB, and configuration file (as handle to the SM).
As written before, be aware, that Simulink can only return or view simulation results, if an “Out“ block or a “Scope“ block is included for the data to view.

Simulating the created model A Simulink SM is a .m file which can be executed in Matlab. The MB (either one or more .slx file(s) or a directory with FMUs) needs to be placed in the same directory as the .m file. On execution of the .m file in Matlab a Simulink model is created, which can be executed then.
In OpenModelica or Dymola an SM is a .mo file which can be executed in the respective simulation program. The MB (either one or more .mo file(s) or a directory containing subdirectories with .mo files (imported preconfigured FMUs)) needs to be loaded. Every FMU imported as .mo file needs to be loaded. The SM has no graphical representation and can only be viewed in text mode.

Building Interface There is an interface to the build method.
The build method can be started from the command line calling SESMoPy with the -b option as seen in Figure 2.2. The input FPES .jsonsestree file is specified.
Remember, that the file(s) containing the MB shall be in the same directory as the FPES file.

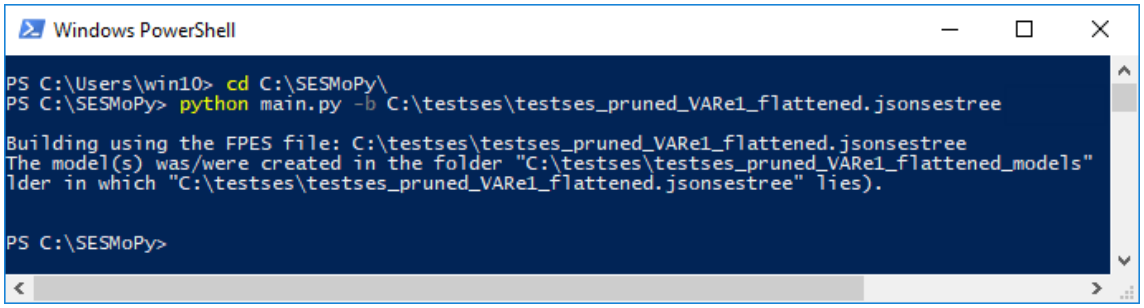

Figure 2.2: Building interface of SESMoPy.

2.4 Building an Executable / Runnable of SESMoPy in Windows and Linux

For Windows and Linux there is the possibility to build an executable / runnable of the program. There is no installation of Python3 / PyQt5 needed on machines for executing these compiled files.

Executable in Windows In the main program directory of SESMoPy is the file *CreateRunnableWindows.cmd*. This file is a Windows-Command script to execute in order to compile an .exe file of this program. Please note, that the executable is dependent of the processor architecture and the operating system. It is not platform independent anymore. A folder containing the .exe file and all for the execution necessary libraries is created. Compilation is done with cx_Freeze 4.3.4. The program cx_Freeze needs a configuration file. This configuration is stored in the file *setup-windows.py* the same directory as the file *CreateRunnableWindows.cmd* is located.

Runnable in Linux In the main program directory of SESMoPy is the file *CreateRunnableLinux.sh*. This file is a Linux Shellscript to execute in order to compile a runnable file of this program. Please note, that the runnable is dependent of the processor architecture and the operating system. It is not platform independent anymore. A .tar.gz archive containing the runnable file and all for the execution necessary libraries is created. Compilation is done with PyInstaller.

3 Background: Concepts for modelbuilding from an SES for different simulators

In this chapter the concepts for modelbuilding from an SES for different simulators are discussed.

As stated before the SES shall be independent of the used simulator. Furthermore the issues were described in Section 2.2: Basic models (blocks) of different simulators have (i) different portnames and (ii) different parameters and parameternames (which may need a different syntax to be configured or parameters with the same name have a different meaning). In order to tackle these issues there are some concepts discussed in the next paragraphs.

Different portnames This can be tackled by placing each basic model in the MB in a subsystem, e.g. in Simulink. See example #01 in Section 4.1 for details.

Different parameters approach #1: Preconfigured MB Place all basic models preconfigured in an MB. This approach leads to many basic models in the MB. The MB will be hard and costly to maintain. Furthermore some simulators need special basic models, e.g. an "Out" block for Simulink to return results. For each simulator one MB needs to be created. SESMoPy *supports this approach* of course, since no parameters need to be set.

Different parameters approach #2: Parameter adaption in the modelbuilder Set the correct parameters and its syntax for the configuration of basic models in the modelbuilder. In this approach much simulator specific code is set in the modelbuilder. For each simulator one MB needs to be created. SESMoPy *does not support this approach.*

Different parameters approach #3: MB with an additional function Place an additional function to the MB for setting the parameters in the correct syntax to configure the basic models. This approach leads to less simulator specific code in the modelbuilder, but the additional function for configuration of the MB is costly to program for complex models. Any additional blocks needed for a simulator (e.g. an "Out" block for Simulink to return results) can be added, configured and connected in this function as well. For each simulator one MB needs to be created. See example #02 in Section 4.2 for details. SESMoPy *supports this approach.*

Different parameters approach #4: Using OpenModelica and FMI for model exchange (export whole model as FMU) In this approach OpenModelica and the simulator independent standard interface FMI (for model exchange) are used. An MB for OpenModelica is created and the ports as well as the configuration of the blocks in the FPES are defined in OpenModelica syntax. The native OpenModelica model is built and exported as FMU. Finally the (configured) model FMU is imported in the chosen simulator. Many simulators support FMI and an API is implemented in these simulators. The simulator specific API needs to be implemented in SESMoPy, so a simulator specific model based on the model FMU can be created. However, there is no need for simulator specific variables and syntax, since a configured FMU is loaded in the simulator. This approach only needs one modelbase (created in OpenModelica) for all simulators. There is no possibility of changing the structure of the model built, only parameters can be set. SESMoPy *does not support this approach.*

Different parameters approach #5: Using OpenModelica and FMI for model exchange (export block FMUs) This approach is nearly the same as before. OpenModelica and the simulator independent standard interface FMI (for model exchange) are used. An MB for OpenModelica is created and the configuration of the blocks in the FPES are defined in OpenModelica syntax. Each OpenModelica block is configured according to the information in the FPES and the configured OpenModelica native block is exported as FMU. Finally the (configured) FMU is imported in the chosen simulator. In the couplings the ports defined by the FMUs need to be defined. Many simulators support FMI and an API is implemented in these simulators. The simulator specific API needs to be implemented in SESMoPy, so a simulator specific model based on the model FMU can be created. However, there is no need for simulator specific variables and syntax, since a con-

figured FMU is loaded in the simulator. This approach only needs one modelbase (created in OpenModelica) for all simulators. SESMoPy *supports this approach.*

Different parameters approach #6: Using FMI for model exchange and a simulator API In this approach the simulator independent standard interface FMI (for model exchange) is used. From any simulator the needed FMUs (=basic models) are exported for model exchange. The FMUs are configured, the parameters and the syntax are defined by the FMI they implement. Finally the preconfigured FMUs are imported in the chosen simulator. Many simulators support FMI and an API is implemented in these simulators. The simulator specific API needs to be implemented in SESMoPy, so a simulator specific model based on FMUs can be created. However, there is no need for simulator specific variables and syntax, since preconfigured FMUs are loaded in the simulator. The FMUs are collected in one folder, which can be seen as modelbase. Therefore it can be stated that this approach only needs one modelbase for all supported simulators. Processing the FMUs is very slow. See example #03 in Section 4.3 for details. SESMoPy *does not support this approach.*

Different parameters approach #7: Using FMI for co-simulation From any simulator the needed FMUs (=basic models) are exported for co-simulation. These FMUs integrate the needed solver. The FMUs are configured, the parameters and the syntax are defined by the FMI they implement. Using the tools PyFMI [AkF16], [Mod19] or FMPy [CS19] the simulation can be done. There is no need for a simulator specific API. Like in the approach before, only one MB is needed. This approach describes the modelbuilding as well as the simulation step. SESMoPy *does not support this approach.*

4 Examples

The first and second examples are taken from the documentation of SESToPy. The third example demonstrates the use of FMI. The fourth example shows how an EF can be applied to generate different system configurations.
These examples are released with SESMoPy in the JSON format and the MBs for the supported simulators are given. They are stored in the program directory of SESMoPy in a folder called “Examples“.

4.1 Example #01: Simple Model - Aspect Node

This is a basic example showing how to use subsystems for basic models in different modelbases to have the same portinterface. A model based on Example #01 in the documentation of SESToPy is created. However, only one coupling is defined. This is presented in Figure 4.1. In the MB basic models of the types B and C are placed. For the purpose of demonstration in the MB files model type B and type C are blocks from the respective simulator specific libraries. In the Simulink MB the blocks are placed in subsystems in order to have the same portnames as the blocks in the OpenModelica and Dymola MBs have. This example FPES does not show the node defining the simulator. In the .jsonsestree file of this example it needs to be defined in order for functionality of this example. In Section 2.3 this node is introduced.

Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola). A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well.

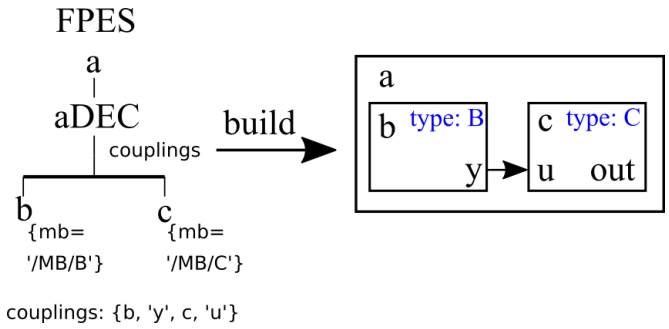

Figure 4.1: FPES and model of Example #01.

4.2 Example #02: Feedback Control System with Optional Feedforward Control

This example was introduced in the documentation of SESToPy. In this example is demonstrated how a model can be generated for different simulators using the same FPES and the native interface. In the documentation of SESToPy two possible FPES of this system are given. However, for the purpose to demonstrate model generation for multiple simulators one FPES is selected. The FPES is shown in Figure 4.2. This example FPES does not show the nodes defining the simulator and the interface as well as the node defining parameter variations for the PID controller. In the .jsonsestree file of this example they need to be defined for functionality of this example. In Section 2.3 these nodes are introduced.

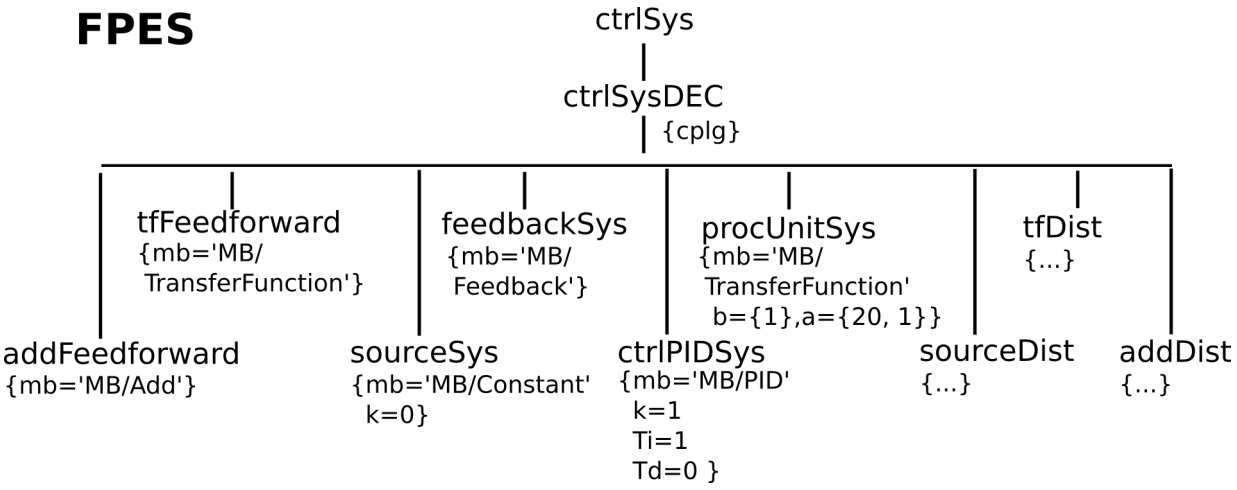

Figure 4.2: FPES for the feedback control system with a feedforward control.

In this example the attributes of nodes in the FPES configuring basic models (blocks) as well as couplings (connecting block ports) are defined for OpenModelica in Open-

Modelica syntax. For example the node *procUnitSys* refers to a block *TransferFunction*, which has the parameters “a“ and “b“. The parameter “a“ is parameterized with “{20,1}“, while “b“ is parameterized with “{1}“.

Couplings connect ports “u“ (inports of a block) and “y“ (outports of a block). In Figure 4.3 the couplings are shown. Notice, that for the couplings the source is the outport of a block and the sink is the inport of a block.

source	port	sink	port
sourceSys	y	feedbackSys	u1
feedbackSys	y	ctrlPIDSys	u
procUnitSys	y	addDist	u2
addDist	y	feedbackSys	u2
sourceDist	y	tfDist	u
tfDist	y	addDist	u1
addFeedforwarc	y	procUnitSys	u
sourceDist	y	tfFeedforward	u
tfFeedforward	y	addFeedforward	u1
ctrlPIDSys	y	addFeedforward	u2

Figure 4.3: Couplings of the feedback control system with a feedforward control.

These names and syntax are applicable to OpenModelica and Dymola, but not to Simulink. Therefore the Simulink MB is adapted. In the MB each block is placed under a subsystem (“Create Subsystem from Selection“). In the subsystem it gets inport and/or outports, which are named “y“ or “u“. No other blocks in the subsystem may have the portnames “y“ or “u“, e.g. no MatlabFunction blocks. Furthermore for attributes configuring a block a MATLAB function to interprete the values and transform them in MATLAB syntax is written. As written before, when building a Simulink model the parameters are written as variables in the script created in the build process. In the MATLAB function these variables are interpreted and the respective block is configured. This function is called before beginning a simulation run. Therefore it is called in Simulink’s “InitFcn“ (Open Simulink MB.slx -> make sure you are on the top layer, not in a subsystem -> View -> Property Inspector -> Tab "Properties" -> Callbacks -> InitFcn).

Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola).
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well.

4.3 Example #03: Feedback Control System with Optional Feedforward Control - FMI

This example is the same as example #02, only FMI is used. In this example is demonstrated how a model can be generated for different simulators using the same FPES and FMI. The FPES is shown in Figure 4.2. It is the same as example #02.

This example FPES does not show the nodes defining the simulator and the interface as well as the node defining parameter variations for the PID controller. In the .jsonsestree file of this example they need to be defined for functionality of this example. In Section 2.3 these nodes are introduced.

In this example the attributes of nodes in the FPES configuring basic models (blocks) as well as block ports are defined for OpenModelica. For example the node *procUnitSys* refers to a block *TransferFunction*, which has the parameters “a“ and “b“. The parameter “a“ is parameterized with “{20,1}“, while “b“ is parameterized with “{1}“.

Couplings connect ports “u“ (inports of a block) and “y“ (outports of a block). In Figure 4.3 the couplings are shown. Notice, that for the couplings the source is the outport of a block and the sink is the inport of a block.

In the build process the OpenModelica blocks are configured and FMUs created. These FMUs can be imported in any simulator supporting FMI.
Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola). In the build process the FMUs are configured and depending on the chosen simulator imported in the simulator.
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well. That means going into the MB directory and every subdirectory to load the imported FMU (the .mo file).

5 Related Work

For an introduction to the development please look at the documentation of the SES modeling tool SESToPy.

EFs were described for the System-Model-Simulator perspective as the environment surrounding the model in [ZKP00]. This concept was extended by [TM06]. In [DKM+17] the authors define requirements for EFs and show....

Bibliography

[AkF16] Christian Andersson, Johan Åkesson, and Claus Führer. Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface. Technical Report 2, Centre for Mathematical Sciences, Lund University, 2016.

[CS19] CATIA-Systems. Fmpy github. https://github.com/CATIA-Systems/FMPy, 2019.

[DKM+17] Joachim Denil, Stefan Klikovits, Pieter J. Mosterman, Antonio Vallecillo, and Hans Vangheluwe. The experiment model and validity frame in m&s. In *Proceedings of the Symposium on Theory of Modeling & Simulation*, TMS/DEVS '17, pages 10:1–10:12, San Diego, CA, USA, 2017. Society for Computer Simulation International. URL: http://dl.acm.org/citation.cfm?id=3108905.3108915.

[Mod19] Modelon. Pyfmi github. https://github.com/CATIA-Systems/FMPy, 2019.

[TM06] Mamadou K. Traoré and Alexandre Muzy. Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory*, 14(2):126–142, 2006. URL: https://doi.org/10.1016/j.simpat.2005.03.002, doi:10.1016/j.simpat.2005.03.002.

[ZKP00] B.P. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Elsevier Science, 2000. URL: https://books.google.de/books?id=REzmYOQmHuQC.

List of Figures

1.1 Extended SES/MB-based infrastructure. 4
1.2 Part of the eSES/MB infrastructure supported by SESMoPy. 4

2.1 SES/MB supported by SESMoPy. 9
2.2 Building interface of SESMoPy. 13

4.1 FPES and model of Example #01. 18
4.2 FPES for the feedback control system with a feedforward control. . . 18
4.3 Couplings of the feedback control system with a feedforward control. 19

List of Tables

2.1 Entries in the configuration file. 11

List of Listings

Acronyms

CEA Computational Engineering und Automation.

EF Experimental Frame.

eSES/MB extended SES/MB.

FMI Functional Mock-up Interface.

FMU Functional Mock-up Unit.

FPES Flattened Pruned Entity Structure.

MB Model Base.

SES System Entity Structure.

SM simulation models.