Documentation

SESMoPy - SES Modelbuilder Python

A Python Modelbuilder Using an SES and an MB

as of: September 17, 2020

by: Research Group CEA

Contents

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. Currently there are no more modules needed than given in the Python Standard Library. As simulation softwares Matlab R2018a (for Simulink), OpenModelica 1.12.0 with the Modelica library version 3.2.2, and Dymola 2018 are used.

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.

In Figure 1.2 it can be seen that one FPES can lead to several simulation models (SM). One FPES represents one structure variant of a system. The FPES can encode several system configurations. In the FPES (derived from an SES) a range of values
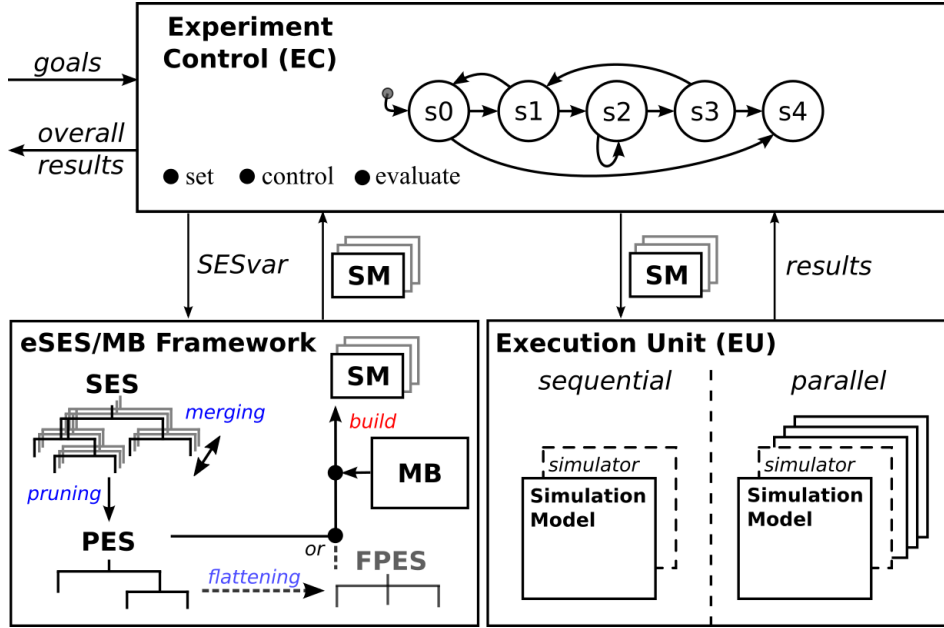
Figure 1.1: Extended SES/MB-based infrastructure.
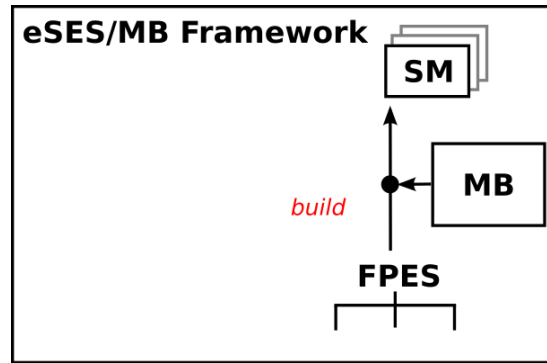
Figure 1.2: Part of the eSES/MB infrastructure supported by SESMoPy.

can be defined for attributes of entity nodes. For an entity node, that refers to a basic model, the attributes are linked to the parameters of the basic model. A range of values is thus defined for the parameters of the basic models. Different parameterization leads to a number of SM of the same structure variant.

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 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 ex-

change. 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.

*The native approach is **only** developed for and tested with signal-flow oriented models. Please use the FMI approach. However, in this documentation both approaches are described.*

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.

The SES/MB framework is usually used for simulation-specific problems. However, it can be used for non-simulation-specific component-based systems as well. In the example in Section 4.6 it is shown how SESMoPy can be used for a non-simulation-specific component-based system on the example of building software. The next chapters are not of interest for this non-simulation-specific use.

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 an MB needs to be created. Using the native interface the MB is specific for the simulator. For each simulator one MB needs to be created. Using FMI the same MB can be used without adaption with several simulators (since the simulators support FMUs defining the FMI). An OpenModelica modelbase can be created and/or FMUs exported from any simulation program can be used as basic models in connection with FMI. Furthermore the connection to the SES is specified and the build method is 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.

Furthermore it is shown how to export an FMU using OpenModelica. 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. A directory containing a number of FMU files is called FMU MB in this context.

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. The inports and/or outports of the created subsystem are named then. See examples in Sections 4.1 and 4.2 for details. Save it with the name, the MB shall get. It is saved as a .slx file.

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! A block can also be entered as sourcecode. In that case the block the "Extends (optional)" field is left blank. 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. It is saved as a .mo file.

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. It is saved as a .mo file.

FMU export on the example of OpenModelica Open OpenModelica Connection Editor (OMEdit). Go to Tools -> Options -> FMI -> Type and select "Model Exchange". On the left side in the Libraries Navigator go in "Modelica" and select a block. Make a rightclick on the block and choose "Export FMU". It takes some time until the process is finished. In the messagewindow in the lower center part of the screen it is printed where the FMU is saved.

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 and discussed in this section. The

SES shall be independent of the simulator the model is built for.

For usage in different simulators *using the native interface*, one modelbase for each simulator needs to be created. The basic models for different simulators need to have the same interface (same parameters, same ports) to be usable with 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 consisting of input blocks, the block itself, and output blocks 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. Simulator specific blocks can be added by this function as well. 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*, one modelbase for all simulators is created. In the SES the attribute with the name "mb" refers to (i) an FMU or to (ii) a block in an OpenModelica MB. According to the instructions in the FPES (blocks, parameterization, couplings) an OpenModelica model is built. All FMUs are imported in OpenModelica using an API function. Thus all values for configuration of basic models (blocks) need to be defined in OpenModelica syntax in the SES attributes referring to parameters of a block. See the files of the example #03 in Section 4.3 for details.

In the SES it needs to be defined which simulator and interface to use.

MB attribute A leaf node gets an attribute with the name "mb" and a value like "modelbasename/blockname". The value needs to be in quotation marks.
For the *native interface* "modelbasename" is the same name as the filename of the file(s) containing the MB. Depending on the simulator "modelbasename" refers to the file "modelbasename.slx" for Simulink and to "modelbasename.mo" for Open-Modelica and Dymola. For each simulator a dedicated simulator specific modelbase with the same name is needed. In the MB attribute in the SES the ending of the simulator specific MB file is not set, it is appended in the modelbuilder depending on the used simulator. The "blockname" is the name of a block in a simulator specific MB "modelbasename". A whole path with folders starting from the directory containing the FPES file can be set.
For FMI there are two possibilities: An MB attribute can (i) refer to an FMU or it can (ii) refer to an OpenModelica modelbase. Only one modelbase (which can

consist of OpenModelica basic models and FMUs) for each simulator is needed. In case of (i) the modelbasename is the path to the respective FMU starting from the directory containing the FPES file. The blockname is the name of an FMU file. The MB attribute entry is then "modelbasename/block.fmu". In this example the FMU block with the name "block.fmu" lies in a directory named "modelbasename", which is referred to as FMU MB. In case of (ii) the entry in the MB attribute is "modelbase.mo/blockname". The modelbasename refers to the file "modelbase.mo" and the blockname is the name of a block in the OpenModelica MB "modelbase.mo". A whole path with folders starting from the directory containing the FPES file can be set.

Because of the MB attribute the connection between a node and a block (basic model) in the MB can be established without the node being 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 file. In case simulator specific MBs are used, all MBs have to contain blocks (basic models) with the same names that fit the referenced blocks in the MB attributes.

Names of the block parameters Further attributes of entities in the FPES configure a block (basic model). Parameters of a block can be set with attributes of the node, which refers to the block by the MB attribute. 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 for Modelica blocks). 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.

In order to use generalized names for block parameters in the SES, *using the native interface* the modelbases for different simulators need to be adjusted as stated before, e.g. with a function which is defined for the MB and executed before simulation.

Using FMI, in the SES the parameters of a referenced basic model (block) need to fit (i) the FMU or (ii) the OpenModelica block (depending on what the referenced basic model is). FMUs keep their parameter names also when imported in a simulator.

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

Every block has inputs and outputs. When building the model, the ports of blocks need to be connected using the couplings defined in the SES. These are recalculated during pruning and flattening. Couplings consist of the tuple (sourceblock name, sourceport / porttype, sinkblock name, sinkport / porttype).

The portnames and porttypes 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). It is of type "Real".

The porttypes define whether the port referred to in the coupling of a basic model is a Signal Port of type Real, Signal Port of type Integer, Signal Port of type Boolean, or a Physical Port.

For *the native interface*: In order to use generalized portnames in the SES, the modelbases for different simulators need to be adjusted by creating coupled models consisting of input blocks, the block itself, and output blocks as written above.

Using FMI the portnames and porttypes of a basic model need to be defined for OpenModelica in the SES, since the whole model is built and configured in OpenModelica.

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
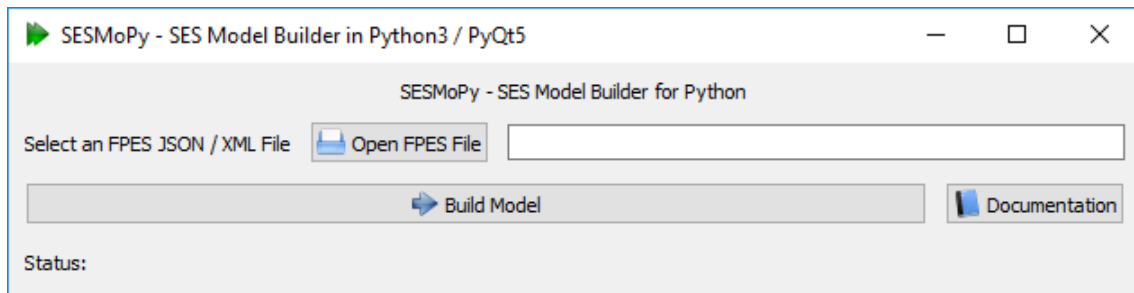

Figure 2.1: SESMoPy's UI.

modelbuilding process is started with click on the button "Build Model". The file(s) containing the MB (which can consist of simulator specific files and/or FMUs) needs 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. In case of FMI the MB can also be a folder / folders containing FMUs.

For modelbuilding in the SES a node defining the simulator in the attributes "SIMU-LATOR" 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 INTER-FACE can be "native" or "FMI".

In the build process for a structure variant (coded in an FPES) a folder (called modelfolder) is created in which the created models (of the same structure variant) are placed. The created models have a different configuration (parameterization). The folder gets the name of the FPES file with an appended "_models".

For Simulink as target simulator Matlab .m scripts can be used to create and configure Simulink .slx models. Thus a model is written as Matlab .m script containing instructions on how to create and manipulate the Simulink .slx model. This script is called Simulation Model Representation (SMR).
For OpenModelica or Dymola as target simulator models are created as textfiles and get the fileending .mo . With scripts of type .mos (MOdelicaScript) the API functions of the target simulator are called e.g. for loading a .mo file or exporting or importing an FMU. The models can only be manipulated (e.g. extended with configuration information) by writing in the text of the .mo file. Since the .mo file contains an executable model, it is called Simulation Model Executable (SME).

2.3.1 Processing using the native interface

For *the native interface* the FPES is analysed and following these information (blocks referenced by the MB attribute, parameterization, couplings) simulator specific textfiles with instructions defining the SM (SME or SMR) are created.
Furthermore the MB is copied in the modelfolder. For 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. For the configuration a function defined for the MB is used, which is run before simulation.

Information on the created SM are written to a configuration file discussed later. A modelfile of an SM gets the name of the of the FPES file it was derived from 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. See example #02 in Section 4.2 for details.

Created SM in Simulink   A Simulink model is created as SMR, a 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. Furthermore there is a command to fill the "InitFcn" of a Simulink model with a function defined for the configuration of blocks (basic models) in the model (see Section 2.2 and example #02 in Section 4.2).

A block is added with the *add_block('modelbase/blocktype', 'modelname/blockname');* function. Attributes configuring block parameters are placed as variables in the script. These variables are named "blockname_parametername". All blocks are inserted. An additional function defined for the modelbase reads the variables and configures the block they are defined for. If Simulink specific blocks are needed, this can also be defined in this function.

In the next part of the file the couplings are defined. Some code is necessary to find the correct portnumber from a given portname to draw a coupling. The couplings are drawn with the *add_line('modelname', outportHandle, inportHandle);* function. All couplings are inserted. See example #02 in Section 4.2.

Created SM in OpenModelica and Dymola   A model for OpenModelica and Dymola is created as SME. It is 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. All blocks are inserted. The definition of the blocks ends with the keyword *equation*.

In the next part of the file the couplings are defined. An entry follows the structure: *connect(blockname.sourceport, blockname.sinkport);* All couplings are inserted.

The model description ends with the keyword *end* and the name of the model. See

example #02 in Section 4.2.

2.3.2 Processing using FMI

Using FMI the FPES is analysed and following these information (blocks referenced by the MB attribute, parameterization, couplings) textfiles defining OpenModelica models are created. The steps for that are given next. However, please be aware that OpenModelica's FMI capabilities are used. OpenModelica's "omc" command needs to be startable from the command window / shell. Thus OpenModelica must be on the users's or system's path. For Windows, the bin folder of OpenModelica needs to be on the path of the user / system. For Linux, "omc" of OpenModelica must be able to be started with the command "omc" from the shell. Otherwise a symbolic link to the omc executable with the name "omc" in the /usr/bin folder needs to be placed with the command:

```
ln -s /path/to/omc /usr/bin/omc
```

The MB is copied in the modelfolder. Using FMI there are two possibilities for defining a basic model in an MB as stated before. (i) An FMU can be referenced or (ii) an OpenModelica MB containing OpenModelica blocks can be referenced in the MB attribute. For both ways the processing is a bit different. In Figure 2.2 the processing for one model configuration is shown. This is described in detail next.
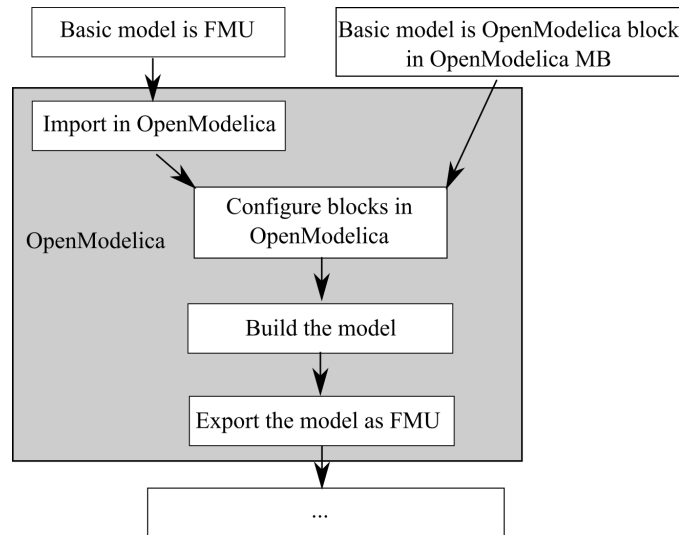

Figure 2.2: Processing of the MB using FMI.

(i): The FMU (=basic model) is tested with the "FMU Compliance Checker" developed by Modelon AB and imported in OpenModelica using an Open-

Modelica API function with a .mos (MOdelicaScript) script. When import-
ing in OpenModelica, a block (.mo file) is created defining the OpenModelica
interface for the FMU. The block is configured. This is done with attributes
in the FPES node with the MB attribute referring to the imported FMU.
They configure the in OpenModelica imported FMU and therefore need to
be in OpenModelica syntax as written before. The configuration is done by
searching for the parameter and inserting the attribute values in the .mo
file which defines the OpenModelica interface of the FMU. Start parameters
for a block are set in the created OpenModelica model in the next step.
(ii): The block (=basic model) in the OpenModelica MB is configured.
Therefore each block from the OpenModelica MB is placed in one .mo file.
The configuration of the block is added (from the attributes in the SES
node with the MB attribute referring to the block). The values of these
attributes need to be in OpenModelica syntax as written before. In case
the block is given in sourcecode, the configuration is done by searching for
the parameter and inserting the attribute values in the OpenModelica .mo
file of the block. If the block is given as "extends" another block, the con-
figuration is done by adding the parameter and its value in brackets at the
end of the line, in which the extension is described. Start parameters for a
block are set in the created OpenModelica model in the next step.

Creating an OpenModelica model and export the model as FMU An
OpenModelica model is created from the blocks (basic models) according to the
information in the FPES.

The configured blocks are added and the start parameters of a block as defined in
the attributes of a node in the FPES are set for each block. The start parameters
include " _start " in their name. Attribute names as defined in the FPES need to be
adapted for OpenModelica, e.g. the _ in a variable name in the FPES is replaced by
(...) in OpenModelica code, if the _ is followed by the word "start". So the attribute
" v_start=1 " in the FPES is replaced with " v(start=1) " in OpenModelica code.
In the OpenModelica user interface this parameter is represented as " v.start ".

For each port of a block connected to another block the porttype is given in the
respective coupling.

The blocks in *signal-flow oriented systems* have signal ports. For each coupling con-
necting signal ports an output block is added and connected. The output block must

fit the porttype specified in the coupling. The output block gets the name *Block-name_Portname_Out*. "Blockname" and "Portname" are taken from the source-block / sourceport in a coupling. An example for that is example #03 in Section 4.3.

The blocks in *physical systems* have physical ports. A physical port is a connection between blocks, which contains several variables. These can be potential variables or flow variables (potential, flow, stream).

For potential variables an output block is added and connected to the potential variable. Each port of a physical block is connected that way (all ports are given in the couplings). The output blocks get the name *Blockname_Portname_Variable_Out*. "Blockname" and "Portname" are taken from the coupling.

Stream variables need an indicator (sensor) integrated in the coupling between physical blocks. These sensors get the current value of the stream variable and their output then is connected to an output block. The sensor blocks get the name *Block-name_Sensorname* and the output blocks get the name *Blockname_Variable_Out*. "Blockname" is taken from the sourceblock in a coupling. Since physical systems of different type have different variables in their connectors, different sensors are needed for each type of system. Thus the porttype needs to represent the type of the system. Examples for that are examples #04 and #05 in Sections 4.4 and 4.5.

Possible porttypes are given in Table 2.1. These steps (FMU import if needed, con-

Table 2.1: Possible types of ports of blocks given in the couplings.

Porttype	Meaning
SPR	SignalPortReal
SPI	SignalPortInteger
SPB	SignalPortBoolean
PPEA	PhysicalPortElectricalAnalog
PPMT	PhysicalPortMechanicsTranslational

figuration of the blocks and model creation in OpenModelica) are repeated for all models of one structure variant (coded in one FPES) with different configurations. Finally the configured models are exported as whole model FMUs using an Open-Modelica API function as shown in Figure 2.2. The output blocks in the OpenMod-elica models represent the output ports in the exported model FMUs. The output ports are the interface of the model FMU. The model FMUs get names of the of the FPES file with an appended string _model_ and a number of the model. All model FMUs of one structure variant (coded in one FPES) with different configurations

are in one folder. The created configured FMUs are tested with the "FMU Compliance Checker" developed by Modelon AB. The model FMUs define the standardized interface FMI and are therefore applicable for all supported simulators (not all simulators supporting FMI, since the simulator API for FMI is used).

When using OpenModelica or Dymola as target simulator these model FMUs are imported in the respective simulator and .mo files of the imported FMUs are created. Using Simulink as target simulator the model FMUs are only imported on simulation run.

For importing the model FMUs into OpenModelica and Dymola as target simulators the executable "omc" of OpenModelica / the executable "dymola" of Dymola need to be startable from the command window / shell. Thus OpenModelica / Dymola must be on the user's or system's path. For Windows, the bin folder of OpenModelica needs to be on the path of the user / system. For Dymola this is the bin or bin64 folder. For Linux, "omc" of OpenModelica / Dymola must be able to be started with the command "omc" / "dymola" from the shell. Otherwise a symbolic link to the omc / Dymola executable with the name "omc" / "dymola" in the /usr/bin folder needs to be placed with the commands:

```
ln -s /path/to/omc /usr/bin/omc
ln -s /path/to/dymola /usr/bin/dymola
```

Finally simulator specific textfiles need to be created. In case of Simulink as target simulator these files are SMRs – Matlab .m scripts, which hold instructions on how to build and execute the model FMU in a Simulink model. In case of OpenModelica and Dymola as target simulators these files are SMEs – model .mo files and just hold information how to instantiate the imported model FMU. Together with the model FMU these files are referred to as SM. Figure 2.2 is updated in Figure 2.3 – again for a single model.

Information on the created SM are written to a configuration file discussed later. A simulator specific modelfile of an SM gets the name of the of the FPES file it was derived from with an appended string _model_ and a number of the model. All SMs with different configurations, but of one structure variant (coded in one FPES) are in one folder. See example #03 in Section 4.3 for details.

Created SM in Simulink The simulator specific textfile of the SM is explained here. It is a Matlab script .m file containing instructions on how to create the Simulink model (SMR).

With the first commands in the file a new Simulink model is created. Then the
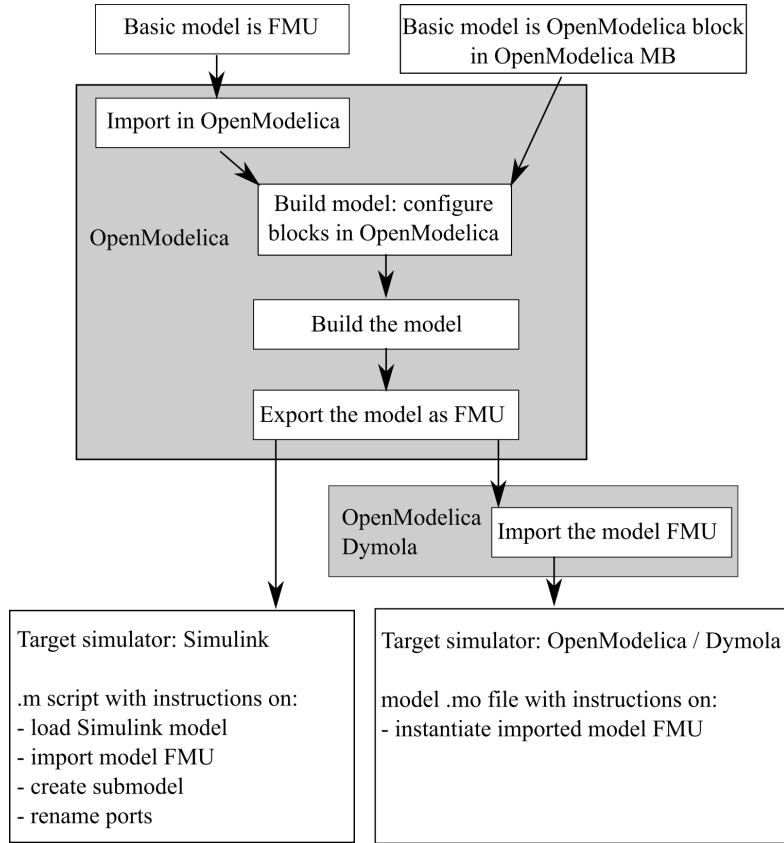
Figure 2.3: Processing of the MB using FMI with the target simulator.

Simulink FMU block is added and configured with the model FMU of the SM. Simulink imports and extracts the FMU and a submodel of the FMU block is created. This submodel is named like the model FMU. Creating a submodel is done to get named input and output ports. The ports are just named "In1", "In2"... and "Out1", "Out2"... first. The next instructions are about finding the inputs and outputs of the FMU in order to get their portnumber and rename them according to the signal they are connected to. Therefore there are instructions to parse the "modelDescription.xml" file of the extracted FMU.

Created SM in OpenModelica and Dymola The simulator specific textfile of the SM is explained here. It is an OpenModelica / Dymola model .mo file containing information on instantiating the imported model FMU (SME). Remember, that for OpenModelica / Dymola the model FMUs are already imported in the target simulator.

It begins with the keyword *model* and the name of the model. The name of the model is the name of the imported model FMU. A block with the imported FMU as

type and name with an appended "1" is instantiated. There are no couplings, since the whole model FMU is instantiated. Therefore the *equation* section is empty. The model description ends with the keyword *end* and the name of the model.

2.3.3 Configuration file

For every structure variant a configuration file "conf.txt" with the information according to Table 2.2 is written in the modelfolder. The configuration file holds basic information on the SM. The first word of every line indicates the information given in the line.

Table 2.2: Entries in the configuration file.

Variable	Description
SESVAR	Assignment of the SES variables in the FPES
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

a) An entry of type SESVAR indicates the assignment of the SES variables in the FPES the model is built with. If the FPES has no SES variables, there is no entry of this type. This entry has no influence on the build process. It is for information on the created model only. Please see the documentation of SESToPy for a thorough explanation of SES variables.

b) An entry of type MODELNAMEPARAM indicates one generated model's parameterization.

c) An entry of type MODEL refers to one simulator specific model in an SM including the absolute path, e.g. ("/Path/to/Model.m" or "/Path/to/Model.mo").

d) In the entry SIMULATOR the simulator to use is specified.

e) The entry INTERFACE indicates, whether the "native" interface or "FMI" is used.

f) An entry of type MODELBASE refers to: (i) An MB file with native basic models for the "native" interface. (ii) Using "FMI" this entry has two parts: In the first part is printed in brackets the simulator specific MODEL the model FMU in this MODELBASE entry belongs to. The second part refers to the model FMU implementing the FMI or to an imported model FMU.

There can be several entries SESVAR. For every SES variable in the FPES there is one entry.

For *the native interface* there can be several entries "MODELNAMEPARAM", "MODEL", and "MODELBASE", since there can be several models of the same structure variant with varying parameterization, each of which uses basic models from different modelbases.

For FMI there can be several entries "MODELNAMEPARAM", "MODEL", and "MODELBASE", since there can be several models of the same structure variant with varying parameterization. Each model has an own model FMU.

The configuration file can be extended.

2.3.4 Return value

SESMoPy returns the filepath of the modelfolder containing modelfile, MB, and configuration file. The modelfolder is the handle to the SM.

2.4 Simulating the created model

Using the eSES/MB infrastructure the simulation step is not part of SESMoPy and is tackled in the execution unit SESEuPy. However, the simulation tools can be started manually with the created models:

A Simulink SM is an SMR, a .m file which can be executed in Matlab. The MB (*native interface:* one or more .slx file(s), *FMI:* the respective model FMU) must be placed in the same directory as the SM .m file. This should already be done by SESMoPy. On execution of the .m file in Matlab a Simulink model is created, which can be executed then. "Scope" blocks need to be added and connected to signals of interest in order to view simulation results.
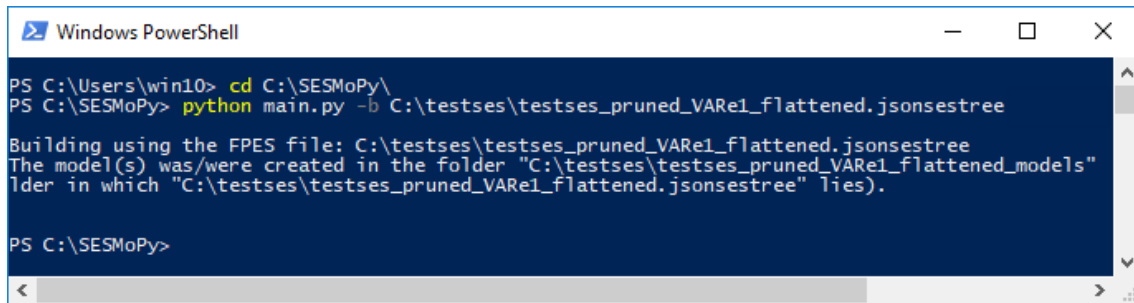
In OpenModelica or Dymola an SM is an SME, a .mo file which can be executed in the respective simulation program. The SM and the MB (*native interface:* one or more .mo file(s) containing a package, *FMI:* the .mo file of the respective imported model FMU) need to be loaded. The SM has no graphical representation and can only be viewed in text mode.

2.5 Build 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 2.4. The input FPES .jsonsestree file is specified. Remember, that the file(s) containing the MB must lie in directories specified in the MB attributes beginning from the directory containing the FPES file.

Figure 2.4: Building interface of SESMoPy.

2.6 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. 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 model-builder   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 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/or FMUs are exported from any simulator. All FMUs are imported in OpenModelica. The ports of couplings 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 target 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 model FMU is loaded into the simulator. This approach only needs one modelbase (consisting of OpenModelica blocks and/or using FMUs) for all simulators. There is no possibility of changing the structure of the model built, only parameters can be set in the built model FMU. Processing FMUs is very slow. SESMoPy *supports 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/or FMUs are exported from any simulator. All FMUs are imported in OpenModelica. The ports as well as the configuration of the blocks in the FPES are defined in OpenModelica syntax. Each block is configured in OpenModelica according to the information in the FPES and the configured OpenModelica block is exported as FMU. Finally all (configured) FMUs are imported in the target simulator and connected according to the couplings in order to build the model. In the couplings the ports specified

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 configured FMU is loaded in the simulator. This approach only needs one modelbase (consisting of OpenModelica blocks and/or using FMUs) for all simulators. Processing FMUs is very slow. SESMoPy *does not support this approach.*

Different parameters approach #6: 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 FMI approaches 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, second, and third example are taken from the documentation of SESToPy. The third example demonstrates the use of FMI for the second example. While these first examples show signal-flow oriented systems, the fourth and fifth example show physical systems. The sixth example shows how to build software using SESMoPy. This example is part of the documentation of SESToPy as well.

All examples are released with SESMoPy in the JSON format as FPES and the MBs for the supported simulators are given. They are stored in the program directory of SESMoPy in a folder called "Examples".

These examples will be adapted for the use with an EF. However, for now a node defining the simulator and the interface and a node defining the parameter variation as introduced in Section 2.3 need to be set for model creation with SESMoPy. In the "Examples" directory a template SES with the name "Template_for_SESMoPy" is placed. It defines the nodes needed for model creation with SESMoPy. In this template SES the simulation model can be merged under the node "simModel". Only nodes of the same name can be merged. Therefore the node "simModel" needs to have the same name as the root of the SES that shall be merged with. After merging the SES needs to be pruned and flattened for use with SESMoPy.

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 Simulink only returns values connected to "Out" ports.
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well.
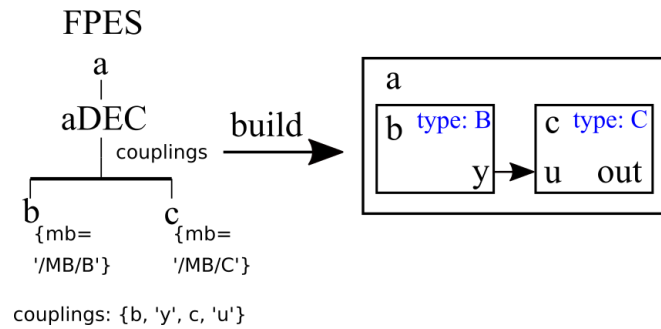

Figure 4.1: FPES and model of Example #01.

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

This example is introduced in the documentation of SESToPy in detail. In this example is demonstrated how a model of a signal-flow oriented system 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 derived. 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. At the beginning of Section 2.3 these nodes are introduced.

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

FPES

```
                                  ctrlSys
                                     |
                                 ctrlSysDEC
                                     | {cplg}
```

| tfFeedforward {mb='MB/ TransferFunction'} | feedbackSys {mb='MB/ Feedback'} | procUnitSys {mb='MB/ TransferFunction' b={1},a={20, 1}} | tfDist {...} |

addFeedforward {mb='MB/Add'}　　sourceSys {mb='MB/Constant' k=0}　　ctrlPIDSys {mb='MB/PID' k=1 Ti=1 Td=0 }　　sourceDist {...}　　addDist {...}
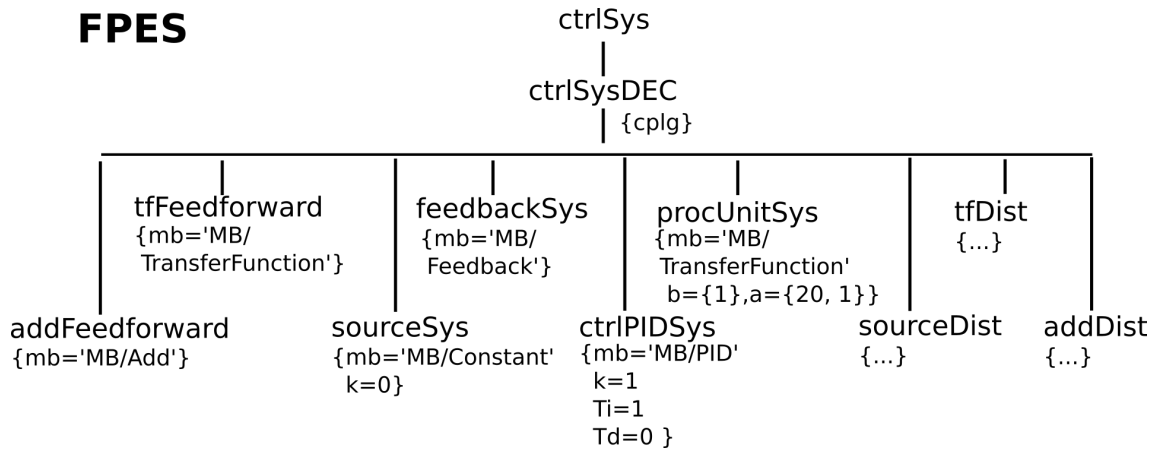
Figure 4.2: FPES for the feedback control system with a feedforward control.

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. Furthermore they all are signal ports of the type "Real" indicated by "SPR".

	source	port name / type	sink	port name / type
1	sourceSys	y / SPR	feedbackSys	u1 / SPR
2	feedbackSys	y / SPR	ctrlPIDSys	u / SPR
3	procUnitSys	y / SPR	addDist	u2 / SPR
4	addDist	y / SPR	feedbackSys	u2 / SPR
5	sourceDist	y / SPR	tfDist	u / SPR
6	tfDist	y / SPR	addDist	u1 / SPR
7	addFeedforward	y / SPR	procUnitSys	u / SPR
8	sourceDist	y / SPR	tfFeedforward	u / SPR
9	tfFeedforward	y / SPR	addFeedforward	u1 / SPR
10	ctrlPIDSys	y / SPR	addFeedforward	u2 / SPR

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. Ports can be tackled like follows: 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

like the respective port in OpenModelica, here "y" or "u". No other blocks in the subsystem may have the portnames "y" or "u", e.g. no MatlabFunction blocks. For attributes configuring a block a Matlab function to interpret 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 SMR, the script created in the build process. In the Matlab function these variables are interpreted and the respective block is configured. This Matlab function is just a template and needs to be adapted for other systems. This function is called before beginning a simulation run. Therefore it is called in Simulink's "InitFcn" (Open Simulink createdModel.slx -> make sure you are on the top layer, not in a subsystem -> View -> Property Inspector -> Tab "Properties" -> Callbacks -> InitFcn). This entry is made during creation of a model.

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 Simulink only returns values connected to "Out" ports.
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 of a signal-flow oriented system can be generated for different simulators using the same FPES and FMI. The FPES is shown in Figure 4.4. It is the same as example #02, only the MB attributes are different.

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.

There are MB attributes in the leaf nodes of the FPES referring to OpenModelica models as well as to FMUs as basic models. All FMUs are imported in OpenModelica. Therefore attributes configuring the basic models (blocks) as well as block ports are defined for OpenModelica. For example the node *procUnitSys* refers to an OpenModelica block *TransferFunction*, which has the parameters "a" and "b". The parameter "a" is parameterized with "{20,1}", while "b" is parameterized with
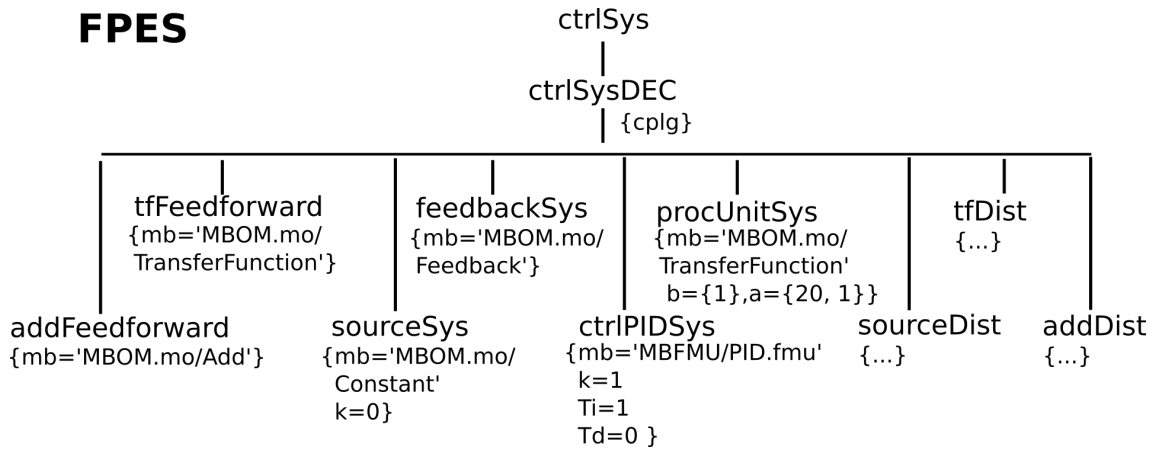
FPES

ctrlSys

ctrlSysDEC
{cplg}

tfFeedforward
{mb='MBOM.mo/
TransferFunction'}

feedbackSys
{mb='MBOM.mo/
Feedback'}

procUnitSys
{mb='MBOM.mo/
TransferFunction'
b={1},a={20, 1}}

tfDist
{...}

addFeedforward
{mb='MBOM.mo/Add'}

sourceSys
{mb='MBOM.mo/
Constant'
k=0}

ctrlPIDSys
{mb='MBFMU/PID.fmu'
k=1
Ti=1
Td=0 }

sourceDist
{...}

addDist
{...}

Figure 4.4: FPES for the feedback control system with a feedforward control - using FMI.

"{1}". An example for a node referring to an FMU is the node *sourceDist*. It refers to *Step.fmu* in the subdirectory of the directory containing the FPES with the name "MBFMU". In the modelbuilding process this FMU is imported in OpenModelica and then the parameter "startTime=0.5"is set.

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. Furthermore they all are signal ports of the type "Real" indicated by "SPR".

In the build process an OpenModelica model for each configuration of the model given in the FPES is built. The blocks in the model are configured using the information in the FPES. Output blocks are added to the models. One created OpenModelica model is presented in Figure 4.5.

These OpenModelica models are exported as FMUs. These model FMUs can be imported in any simulator supporting FMI. However, here a simulator is selected and further steps are taken.
Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola). Furthermore the configured model FMUs are, depending on the target simulator, imported in the simulator.

A model built for Simulink only returns values connected to "Out" ports.
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well. That means loading the imported model FMU (the .mo file).
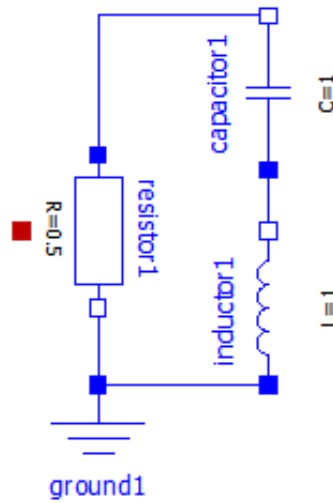
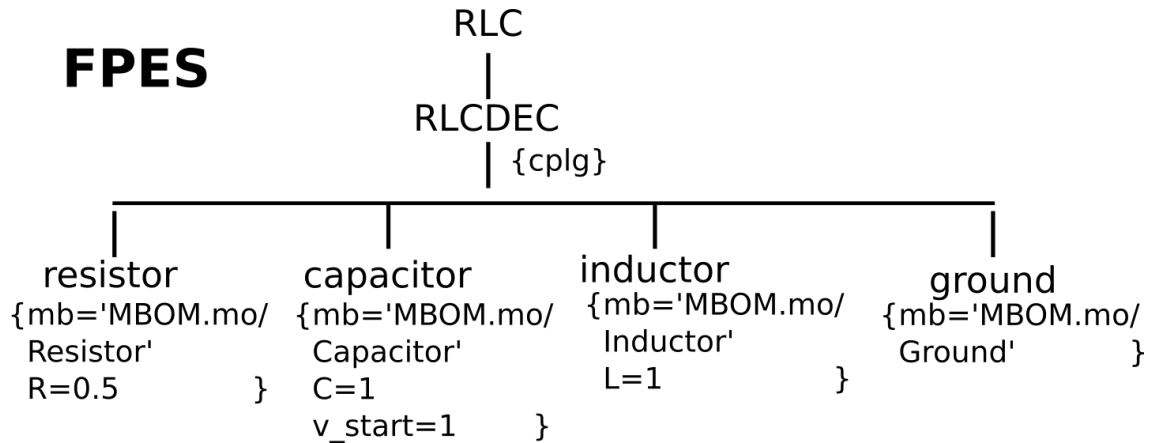**Figure 4.5:** Created OpenModelica model, which will be exported as FMU.

4.4 Example #04: RLC resonant circuit - FMI

This is a typical example of a physical model. It describes a series resonant circuit with a resistor, a capacitor, and an inductor. The capacitor is charged with the voltage $1V$ at the beginning. The resulting oscillation is damped by the resistor. Figure 4.6 shows the circuit.

The FPES used in this example is presented in Figure 4.7.

There are MB attributes in the leaf nodes of the FPES referring to OpenModelica basic models. There could also be FMU basic models as shown in example #03 in Section 4.3. The configuration of the basic models using the attributes of leaf nodes in the FPES is presented in detail in example #03. However, start parameters of a block are set in the created OpenModelica model (see Section 2.3.2). The attribute " v_start=1 " of the capacitor shall be discussed here. In the modelbuilding process a _ is replaced by (...), if the _ is followed by the word "start". So the attribute " v_start=1 " in the FPES is replaced with " v(start=1) " in OpenModelica code. In the OpenModelica user interface this parameter is represented as " v.start ".

Figure 4.6: RLC resonant circuit.

Figure 4.7: FPES for the RLC resonant circuit - using FMI.

The couplings connect the physical ports of type "Pin". Electrical components have a positive pin p and a negative pin n. In Figure 4.8 the couplings are shown. In the couplings the porttype is set as "electrical analog", indicated by "PPEA". This refers to an electrical "Pin" in OpenModelica.

When constructing the couplings, always decide for a direction of flow, do not mix source ports and sink ports.

In the build process an OpenModelica model for each configuration of the model given in the FPES is built (in this example the FPES has only one configuration). The blocks in the model are configured using the information in the FPES.

The "Pin" type of physical ports has the potential variable v (voltage) and the flow variable i (current). In every pin the potential variable v is connected to an output

	source	uid	port name / type	sink	uid	port name / type
1	resistor	4	p / PPEA	capacitor	5	n / PPEA
2	capacitor	5	p / PPEA	inductor	6	n / PPEA
3	inductor	6	p / PPEA	ground	7	p / PPEA
4	ground	7	p / PPEA	resistor	4	n / PPEA

Figure 4.8: Couplings of the RLC resonant circuit.

block. Furthermore, in the coupling between two pins a current sensor is put. A current sensor detects the current i and returns the value. Therefore an output block can be connected to the current sensor. One created OpenModelica model is presented in Figure 4.9. The lines connecting the output blocks of the potential variables are not shown, since only the potential variable is connected.
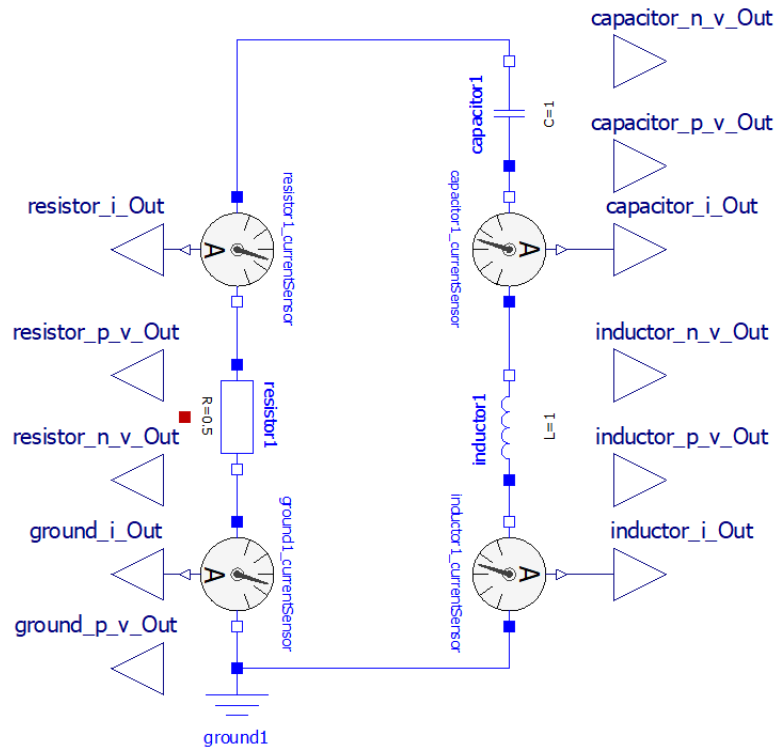
Figure 4.9: Created OpenModelica model, which will be exported as FMU.

These OpenModelica models are exported as FMUs. These model FMUs can be imported in any simulator supporting FMI. However, here a simulator is selected and further steps are taken.

Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola). Furthermore the configured model FMUs are, depending on the target simulator, imported in the simulator.

A model built for Simulink only returns values connected to "Out" ports.
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well. That means loading the imported model FMU (the .mo file).

4.5 Example #05: MSD system - FMI

This is a typical example of a physical model. It is the mechanical analogon to the example #04 in Section 4.4. It describes a mechanical oscillating system consisting of a mass, a spring, and a damper. The mass is connected to a spring and the spring has the length of $1m$ and is stretched to $2m$ length at the beginning. The resulting oscillation is damped by a mechanical damper. Figure 4.10 shows the setup.
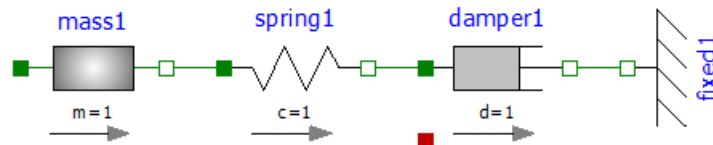
Figure 4.10: MSD setup.

The FPES used in this example is presented in Figure 4.11.
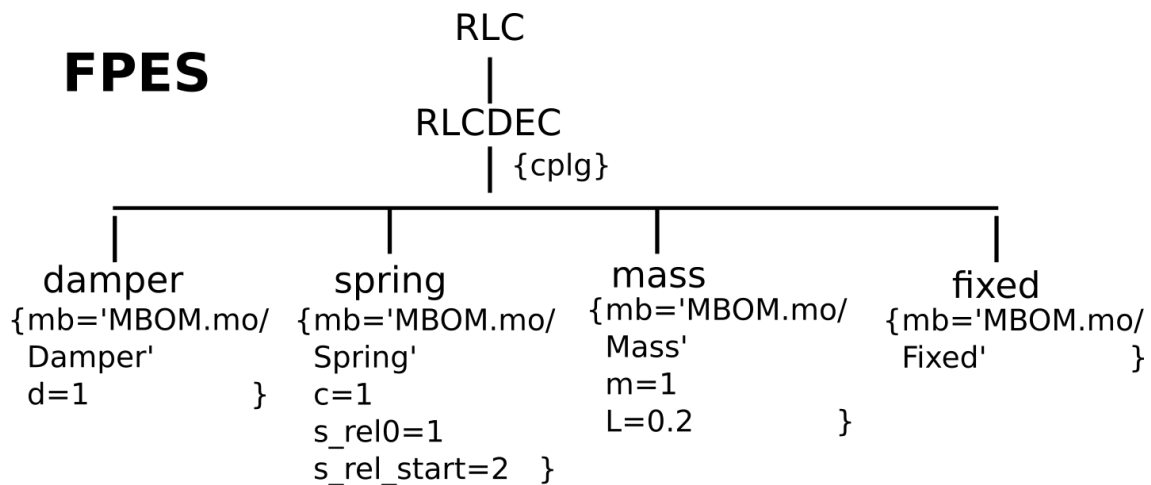
Figure 4.11: FPES for the MSD setup - using FMI.

There are MB attributes in the leaf nodes of the FPES referring to OpenModelica basic models. There could also be FMU basic models as shown in example #03 in Section 4.3. The configuration of the basic models using the attributes of leaf nodes in the FPES is presented in detail in example #03. However, start parameters of a block are set in the created OpenModelica model (see Section 2.3.2). The attribute " s_rel_start=2 " of the spring shall be discussed here. In the modelbuilding process a __ is replaced by (...), if the __ is followed by the word "start". So the attribute " s_rel_start=2 " in the FPES is replaced with " s_rel(start=2) " in OpenModelica code. In the OpenModelica user interface this parameter is represented as " s_rel.start ".

The couplings connect the physical ports of type "Flange". Mechanical translational components have a port "flange_a" and a port "flange_b", or only a "flange" port if they only have one port. In Figure 4.12 the couplings are shown. In the couplings the porttype is set as "mechanics translational", indicated by "PPMT". This refers to an mechanical "Flange port" in OpenModelica.
When constructing the couplings, always decide for a direction of flow, do not mix source ports and sink ports.

	source	uid	port name / type	sink	uid	port name / type
1	mass	4	flange_b / PPMT	spring	5	flange_a / PPMT
2	spring	5	flange_b / PPMT	damper	6	flange_a / PPMT
3	damper	6	flange_b / PPMT	fixed	7	flange / PPMT

Figure 4.12: Couplings of the MSD setup.

In the build process an OpenModelica model for each configuration of the model given in the FPES is built (in this example the FPES has only one configuration). The blocks in the model are configured using the information in the FPES.
The "Flange" type of physical ports has the potential variable s (path) and the flow variable f (force). In every port the potential variable s is connected to an output block. Furthermore, in the coupling between two ports a force sensor is put. A force sensor detects the force f and returns the value. Therefore an output block can be connected to the force sensor. One created OpenModelica model is presented in Figure 4.13. The lines connecting the output blocks of the potential variables are not shown, since only the potential variable is connected.
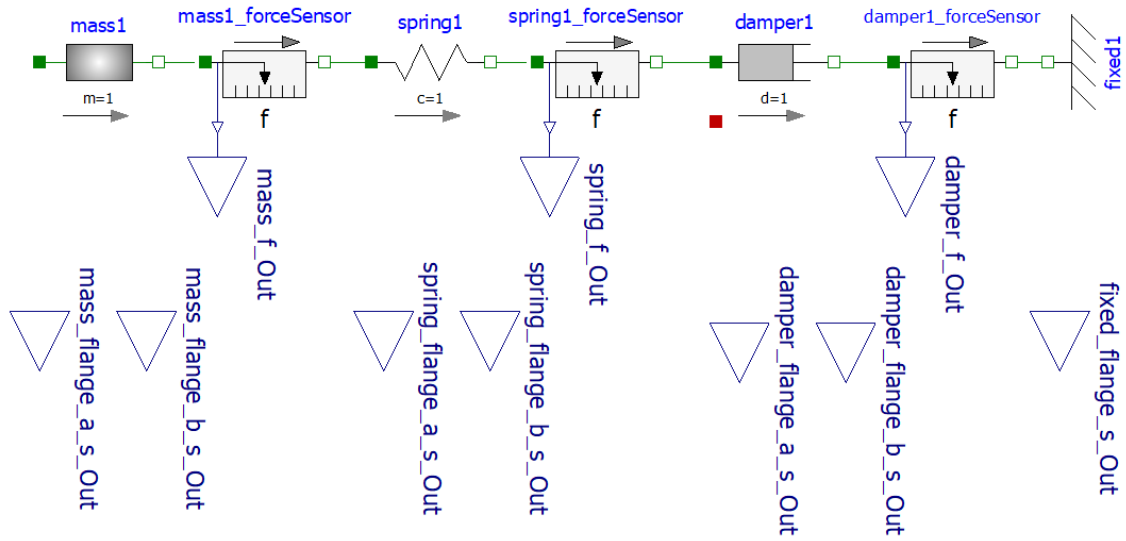
Figure 4.13: Created OpenModelica model, which will be exported as FMU.

These OpenModelica models are exported as FMUs. These model FMUs can be imported in any simulator supporting FMI. However, here a simulator is selected and further steps are taken.

Do not forget in the FPES in the node defining the simulator to specify which simulator to use (Simulink, OpenModelica, Dymola). Furthermore the configured model FMUs are, depending on the target simulator, imported in the simulator.

A model built for Simulink only returns values connected to "Out" ports.
A model built for OpenModelica or Dymola can only be executed when the MB is loaded in OpenModelica or Dymola as well. That means loading the imported model FMU (the .mo file).

4.6 Example #06: Variability in Software: A Clock in HTML / JavaScript

This example is introduced in the documentation of SESToPy. It demonstrates how variants of software can be generated using a HTML / JavaScript clock. The variant of the clock derived for this example shows the date and has a dark style. The derived FPES is presented in Figure 4.14.

This example FPES does not show the node defining the simulator. The "SIMU-LATOR" attribute is set to "None", since it is a non-simulation-specific example.
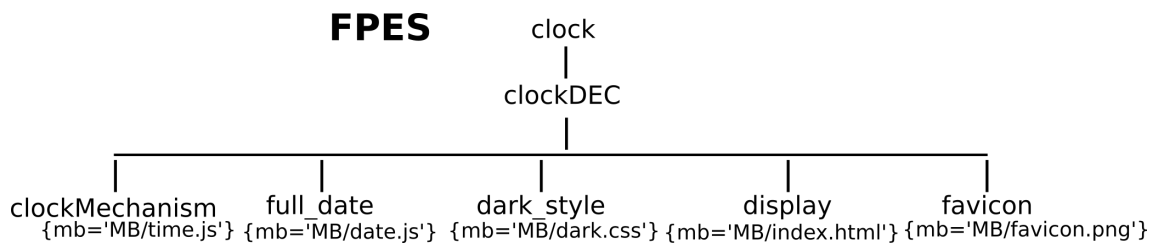
Figure 4.14: FPES of the clock.

In the .jsonsestree file of this example it needs to be defined for functionality of this example. In Section 2.3 this node is introduced.

There are MB attributes in the leaf nodes of the FPES referring to software components, i.e. sourcecode files and pictures. There are no further attributes and there are no couplings.

In the build process the files according to the selected FPES are copied in one folder. Other files are not copied. The time is generated in the file "time.js", the date is generated in the file "date.js", the style is defined in the "dark.css" file. The "index.html" file generates the display and the other files are called in it. An icon for the display is given in the "favicon.png" file.
In the MB there is the file "light.css", which is not copied, since it is not necessary for the selected variant (shown in the FPES).

Furthermore in the MB the Python 3 script "initScript.py" is defined. This script lists the files copied for the selected variant and with these information completes the "index.html" file (which is the central file of the clock). It is executed at the end of the build process after all necessary files for the selected variant are copied. The "initScript.py" can only be given as template, it needs to be adapted for every problem. Figure 4.15 shows the generated software.

This example is only intended for software generation of one program variant, not for use with any simulator.

Figure 4.15: Display of the clock.

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

List of Tables

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.

SME Simulation Model Executable.

SMR Simulation Model Representation.