Documentation

SESMoPy - SES Modelbuilder Python

A Python Modelbuilder Using an SES and an MB

as of: April 2, 2019

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) 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 SES/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). Currently, this software supports the creation of models for the simulators *Simulink*, *OpenModelica* and *Dymola*. Dymola can use the Modelica Model Library.
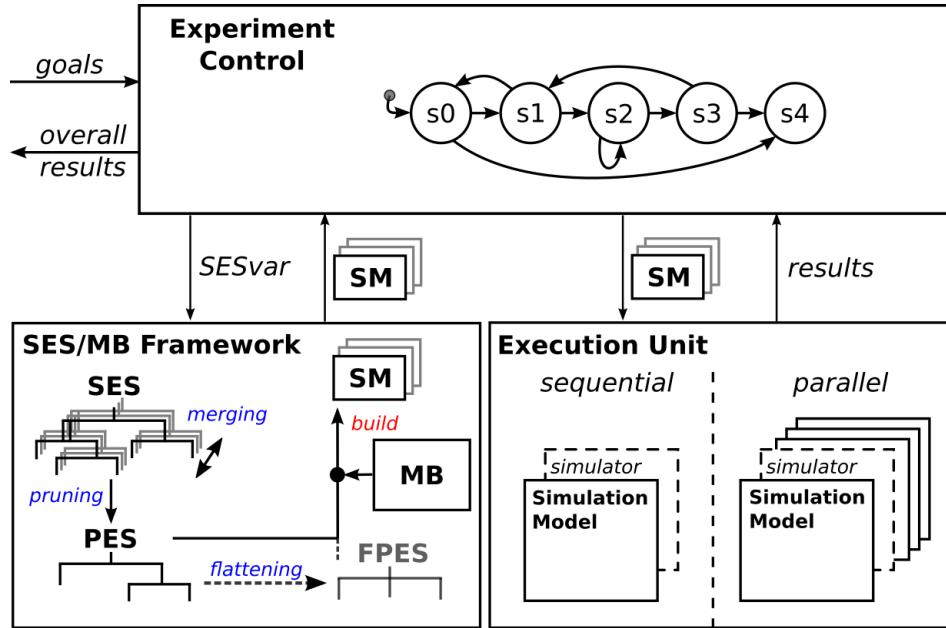
Figure 1.1: Extended SES/MB-based infrastructure.

However, it is planned to support the Functional Mock-up Interface (FMI) as a general interface supported by a number of simulators.
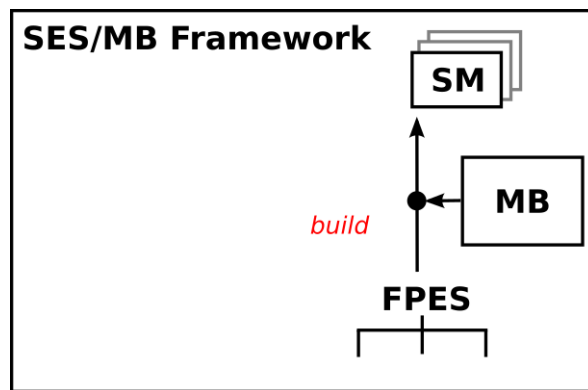
Figure 1.2: SES/MB supported by SESMoPy.

2 Working with SESMoPy

In the following sections of this chapter information on how to use SESMoPy in combination with the different simulation programs shall be given.

As seen in Figure 1.2, there is a need for creating an MB. The MB is specific for the simulator. 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". This section will be extended by generating an FMI modelbase. Functional Mock-up Unit (FMU) explain

Simulink   Open Simulink and create a blank model. Place the blocks needed for the SES in the model and save it. 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.

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.

In Dymola and OpenModelica, 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 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 file containing the MB shall lie in the same directory as 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.
For Simulink an "Out" block needs to be specified in the FPES and added to the MB, in order to view and return simulation results.

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 refering 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, the basic models for different simulators need to

have the same interface (same parameters, same ports) to be used with the same SES. Therefore, the simulator specific modelbases need to be adjusted, e.g. by creating coupled models as one block in the MB so that the basic models for each simulator have the same interface (block parameter names, portnames) and are applicable for the same SES.

However, using FMI solves the need to adjust the blocks in the MB since FMI is an interface supported by several simulators.

Furthermore there is a node to specify the simulator 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 block parameters are defined in the FMU implementing the FMI and is therefore the same for every 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 of blocks are defined in the FMU implementing the FMI and is 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

The FPES is analysed and textfiles with instructions on the SM are created.

For modelbuilding in the SES a node defining the simulator in the attribute "SIM-ULATOR" and a node defining varying parameters as attributes "PARAMVARY" for one structure variant need to be defined. The parameters defined in "PARAM-VARY" are combined. Every combination leads to one SM. The other part of the SES need to define nodes with an MB attribute at their leaves. See examples section! -> Use of Experimental Frame (EF) will change this
Make sure, the file containing the MB lies in the same directory as the .jsonsestree file containing the FPES.

Use of EF to control model generation!

Simulink Beschreibung erstelltes Skript

OpenModelica and Dymola Beschreibung erstelltes Skript

In the build process for every structure variant a folder is created in which models of the same structure variant but different configuration are placed. For every structure variant a configuration file with the information according to Table 2.1 is written. The configuration file holds basic information on the SM. The first word of every line indicates the information given in the line. -> Use of EF will change this

Table 2.1: Entries in the configuration file.

variable	Description
MODEL	Complete path to a model file (*.m or *.mo)
SIMULATOR	"Simulink" or "OpenModelica" or "Dymola"FMI
MODELBASE	Complete path to a modelbase file

There can be several entries "MODEL" and "MODELBASE" since there can be several models, each of which uses basic models from different modelbases.
The configuration file can be extended.

As written before, be aware, that Simulink can only return simulation results, if an "Out" block is included for the data to view.
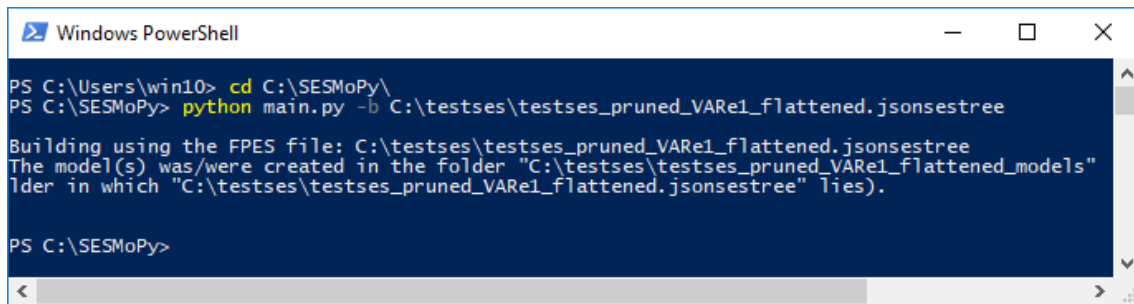
The simulator can be "Simulink", "OpenModelica", or "Dymola". FMI

A folder for the output SM(s) is created in the same directory as the FPES and MB lie, in which the SM are stored together with the configuration file. The folder gets the name of the FPES file with an appended "_models". The model(s) get name(s) of the of the FPES file with an appended string encoding the configuration of the model. So all models with different configurations, but of one structure variant (one FPES) are in one folder. SESMoPy returns the filepath of the created folder.

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 2.1. The input FPES .jsonsestree file is specified.
Remember, that the file containing the MB shall be in the same directory as the FPES file, but this is not necessary for just building the model and not executing it.

Figure 2.1: 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 Examples

THESE EXAMPLES WILL BE CHANGED

The first three examples are taken from the documentation of SESToPy. These examples lack the automatic simulation and evaluation. Therefore there is no EF defined for these models.

The fourth example EXAMPLE WITH EF

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".

3.1 Example #01: Simple Model - Aspect Node

In this example the model according to Example #01 in the documentation of SESToPy is created. This is presented in Figure 3.1. In the MB basic models of the types B and C are placed. For the purpose of demonstration, basic model B is a gain block and basic model C is an integrator block.
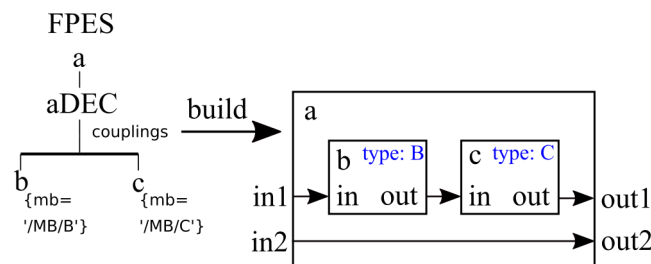

Figure 3.1: FPES and model of Example #01.

3.2 Example #02: Simple Model - Three Nodes

In this example the model according to the flattening example in the documentation of SESToPy is given. This is shown in Figure 3.2. In the MB basic models of the

type mbb, mbc and mbe are placed. For the purpose of demonstration, basic model mbb is an add block, basic model mbc is a gain block, and basic model mbe is an integrator block. It is similar to Example #01 and shall just demonstrate the advantage of the mb-attribute as link to the MB.
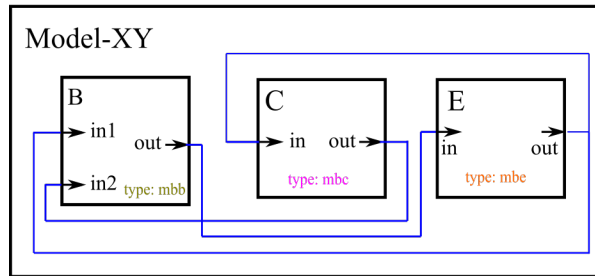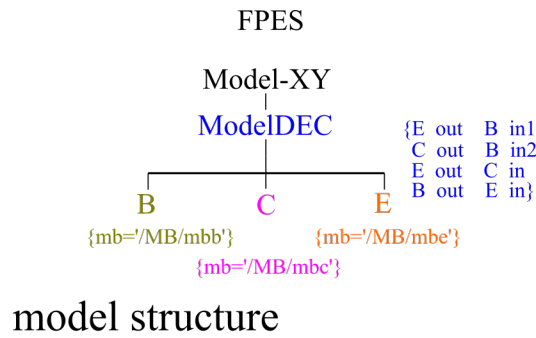
FPES

Model-XY

ModelDEC

{E out   B in1
C out   B in2
E out   C in
B out   E in}

B   C   E

{mb='/MB/mbb'}   {mb='/MB/mbe'}

{mb='/MB/mbc'}

model structure

Model-XY

B
in1
out
in2 type: mbb

C
in   out
type: mbc

E
in
out
type: mbe

Figure 3.2: FPES and model of Example #02.

4 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

[DKM+17] Joachim Denil, Stefan Klikovits, Pieter J. Mosterman, Antonio Val-
lecillo, 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.

[TM06]   Mamadou K. Traoré and Alexandre Muzy. Capturing the dual relation-
ship 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.

FPES Flattened Pruned Entity Structure.

MB Model Base.

SES System Entity Structure.