Documentation

SESEuPy - SES Execution Unit Python

Execution Unit for Models Built with SESMoPy

as of: September 13, 2019

by: Research Group CEA

Contents

1 Introduction

This is the Execution Unit (EU) SESEuPy as part of the System Entity Structure (SES)/Model Base (MB) infrastructure introduced in the documentation of the SES modeling tool SESToPy. It executes simulation model (SM)s built with the software SESMoPy. For information on the SES and its extensions to the extended SES/MB (eSES/MB) infrastructure please read the documentation of SESToPy and how to connect the SES to an MB please read the documentation of SESMoPy. 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 3.4.1, but as of August 2018 it runs in current Python 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 MB for generating models. It was extended to allow automatic model generation and execution of models. The process is depicted in Figure 1.1.

This software provides the execution of SMs created with the modelbuilder software SESMoPy sequential or in parallel. This is shown in Figure 1.2 for clarification.

A detailed explanation on the Python-based software structure supporting the eSES/MB infrastructure is given in the documentation of SESToPy.

SESEuPy has no graphical user interface. The simulators as described in the documentation of the modelbuilder software SESMoPy are supported.

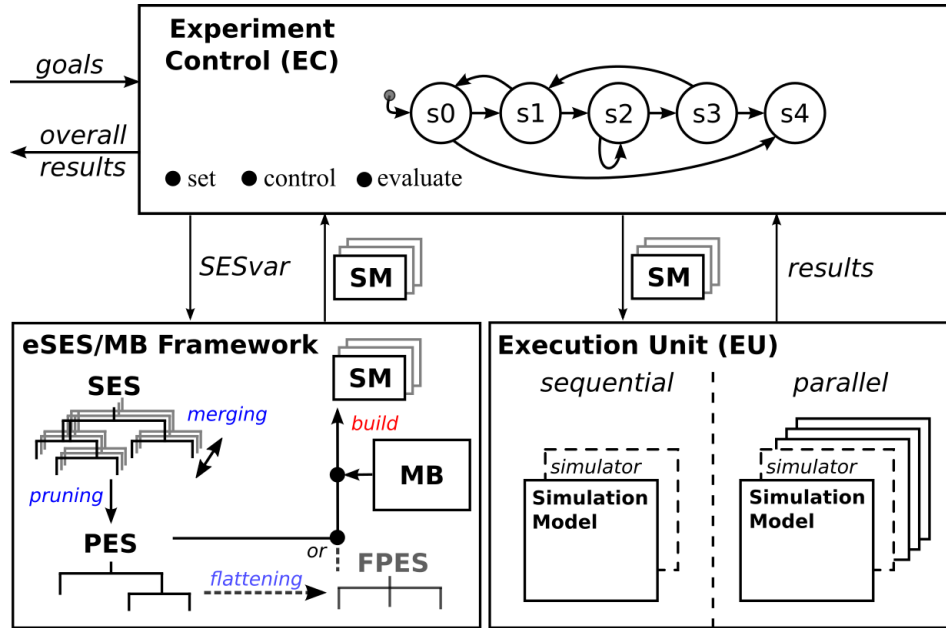In context of a simulator a basic model often is called "block".

Figure 1.1: Extended SES/MB-based infrastructure.
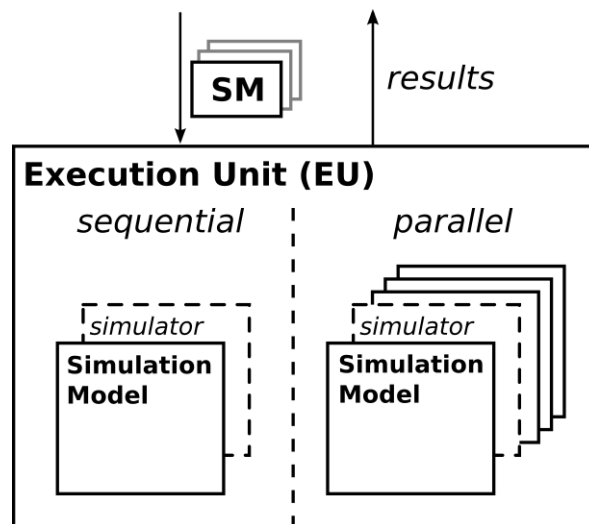

Figure 1.2: Part of the eSES/MB infrastructure supported by SESEuPy.

2 Working with SESEuPy

SESEuPy is used for automatic execution of SMs in the context of the infrastructure given in Figure 1.1. In the following sections of this chapter information on the function of SESEuPy shall be given.

2.1 Simulation folder

As seen in Figures 1.1 and 1.2, SESEuPy receives a number of SMs to execute. Then the SMs can be executed sequential or in parallel. The simulation results are collected and returned.

Since this software is part of the infrastructure, it receives all for the execution necessary data: the models, needed modelbases and a configuration file. These files lie in one directory, in which the simulation takes place – the simulation folder. The configuration file "config.txt" contains absolute links to the model files and modelbase files and contains some simulator information. The configuration file is created by the modelbuilder software SESMoPy and extended with simulator settings in the Experiment Control.

The configuration file needs to have the entries according to Table 2.1. The description of an entry is printed next to an entry in Table 2.1.

a) There can be several entries of the type MODELNAMEPARAM. An entry of this type indicates one generated model's parameterization.

b) Thus there can be several entries of type MODEL. Every entry of this type refers to one model description file including the absolute path, e.g. ("/Path/to/Model.m" or "/Path/to/Model.mo").

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

d) The entry INTERFACE describes the interface a model is specified for. SESEuPy supports the execution of models built by SESMoPy. (i) SESMoPy can build simulator specific "native" models for the supported simulators. They are only applicable for the simulator they are built for and need simulator specific MBs. (ii) SESMoPy can build a whole model as Functional Mock-up Unit (FMU). The FMU implements

Table 2.1: Entries in the configuration file "config.txt".

entry (variable)	Description
MODELNAMEPARAM	Modelname and the model's parameter variation
MODEL	Complete path to a modelfile (*.m or *.mo)
SIMULATOR	"Simulink" or "OpenModelica" or "Dymola"
INTERFACE	"native" or "FMI"
MODELBASE	Complete path to a modelbase file
STARTTIME	The simulation will start at this time.
SOLVER	The solver to use for the simulation, e.g. "ode45" for Simulink or "dassl" for OpenModelica and Dymola.
STOPTIME	The simulation will stop, when the stoptime is reached.
MAXSTEP	Maximum stepwidth for the simulation.
EXECTYPE	Type of the execution: "sequential" or "parallel".
NSIGANA	Names of the signals to analyze in the simulation output: ["Block1.OutputSignal", "Block2.OutputSignal", ...] Notice the quotes!

the general interface Functional Mock-up Interface (FMI). Thus the entry is "FMI". FMUs are applicable for several simulators. However, the entry MODEL refers to a simulator specific model, which loads the model given as FMU. The FMU itself is given as MB. Please see the documentation of SESMoPy for more information on this.

e) There can be several entries of the type MODELBASE. For the native interface every entry refers to a file containing basic models. As described in d) for the native interface these files are simulator specific. Using FMI this entry has two parts: In the first part is printed in brackets the simulator specific model, which the model FMU in this MODELBASE entry belongs to (see d) to understand). The second part refers to the model FMU implementing the FMI – or to a model FMU, which already was imported in SESMoPy in the simulator to use. Which simulators require the import of the model FMU in SESMoPy is given in SESMoPy's documentation.

f) The simulation will start at the STARTTIME.

g) The SOLVER entry defines the solver to use. Different simulators have different solvers.

h) The simulation is executed, until the STOPTIME is reached.

i) The maximum stepwidth for the simulation is set in MAXSTEP. However, some simulators require the number of steps to execute, which can be calculated with the starttime, the stoptime and the maximum stepwidth.

j) The EXECTYPE defines, whether simulations shall be executed sequential or in parallel.

k) In the entry NSIGANA a list with names of signals to analyze is given. Using the native interface in Simulink an "Out" block is added to the ports of output signals of "Block1", "Block2"... in order to receive the simulation results. The blockname and portname to connect an "Out" block is given in NSIGANA. For OpenModelica and Dymola the entry is needed to filter the simulation results of interest. Using FMI the model FMU is created by SESMoPy with an output port at each port of a block connected to another block with a coupling. The output port is called "Blockname_Portname_Out". These ports then can be connected with an "Out" block in Simulink or used to filter simulation results for OpenModelica and Dymola.

There can be more entries in the configuration file than given in Table 2.1, which are not used by SESEuPy and therefore ignored by SESEuPy.

2.2 Performing a Simulation

When performing a simulation run, the configuration file in the folder containing the models to simulate is analyzed. Depending on the simulator to use, there are different steps taken.

Generally   It is important, that the simulation programs are on the user's path, they need to be startable from the command window / shell.
For Windows, the bin folder of Matlab / OpenModelica needs to be on the path of the user / system. For Dymola this is the bin or bin64 folder.
For Linux, Matlab / omc of OpenModelica / Dymola must be able to be started with the command "matlab" / "omc" / "dymola" from the shell. Otherwise a symbolic link to the Matlab / omc / Dymola executable with the name "matlab" / "omc" / "dymola" in the /usr/bin folder needs to be placed.

Simulink   If the simulator is selected as Simulink (in the configuration file coming from the modelbuilder software SESMoPy), the model is given as Simulation Model Representation (SMR). It is an .m script describing the model (generated by SESMoPy), which is extended in SESEuPy as described next. For the native interface Simulink "Out" blocks are added to the model and connected to the respective Simulink model blocks according to the information given in NSIGANA. For FMI Simulink "Out" blocks are added to the model and connected to the ports of the FMU block according to the information given in NSIGANA. The "Out" blocks lets Simulink return a simulation result. A command for the simulation execution with

the simulation information solver, stoptime and maxstep as in Table 2.1 is inserted. Furthermore, there are instructions for generating a CSV file of the simulation results.

Finally, the simulation is executed using a Matlab command calling the extended .m script (SMR). An Simulation Model Executable (SME) is created from the SMR. The SME is a Simulink model.

In case the execution fails, start the created .m script in Matlab.

OpenModelica If the simulator is selected as OpenModelica (in the configuration file coming from the modelbuilder software SESMoPy), the model is given as SME. It is a .mo file. Furthermore a file with the ending .mos (MOdelica Script) is created. This file holds instructions on loading the MB and the model in OpenModelica. Furthermore, simulation information e.g. solver, number of steps (calculated of starttime, stoptime and maxstep) and stoptime as in the configuration file in Table 2.1 are defined. A command to save the simulation results as CSV is applied. The simulation then is executed by calling the created .mos script with the OpenModelica program "omc". By default, the simulation results are printed in the result CSV file. A variable shadowing another variable is not printed in the result CSV file.

The configuration file has the entry NSIGANA, in which the variables (signals) of interest in the simulation results are specified. If a variable of interest is not in the CSV file, it needs to be searched for the variable shadowing it. The information of the connection of variables are given in an XML file "<modelname>_init.xml". This file is parsed in order to find alias variables shadowing a variable of interest. The values of the alias variable are taken and the result CSV file is extended with the name of the variable of interest and the values of the alias variable. Finally variables of interest are extracted and saved in a CSV file with the name of the model.

Please be aware, that for simulation with OpenModelica the name of the model may not start with a number.

In case the execution fails, start the created .mos script from the command line with the command

`omedit scriptname.mos`

to find out the reason.

Dymola If the simulator is selected as Dymola (in the configuration file coming from the modelbuilder software SESMoPy), the model is given as SME. It is a .mo

file. Furthermore a file with the ending .mos (MOdelica Script) is created. This file holds instructions on loading the MB and the model in Dymola. Furthermore, simulation information e.g. solver, number of steps (calculated of starttime, stoptime and maxstep) and stoptime as in the configuration file in Table 2.1 are defined. The simulation then is executed by calling the created .mos script with a Dymola command. By default, Dymola saves the simulation results as .mat file. The tool "alist" lying in the same folder as the Dymola executable is shipped with Dymola, allowing to convert the .mat to CSV files. The signals to convert need to be defined. The configuration file has the entry NSIGANA, in which the signals of interest in the simulation results are specified. These are converted using "alist" and saved in a CSV file with the name of the model.

In case the execution fails, start the created .mos script from the command line with the command

`dymola scriptname.mos`

to find out the reason.

Generally

Using FMI the output signals are named like written in Chapter 2.1 k). For each simulator these signals need to be renamed as the signals in NSIGANA.

The SMs can be called sequentially or in parallel to be simulated in their respective simulation software. Therefore the configuration file in Table 2.1 has the entry EX-ECTYPE, which can have the values "sequential" or "parallel". Currently Dymola SMs cannot be executed in parallel. If EXECTYPE is parallel, for Dymola the models are executed sequentially anyway.

Models cannot be deleted right after simulation using FMI.

When running SESEuPy all models given in the configuration file are simulated with the settings given in the configuration file and a resultfile with simulation results of all models is created in the parent directory of the simulation folder.

2.3 Calling SESEuPy / Execution Unit Interface

The EU can be started from the command line as seen in Figure 2.1.

Remember, that for execution the file containing the MB has to be in the same directory as the generated models by the modelbuilder software SESMoPy. Usually, it should already be placed there by the modelbuilder software SESMoPy when generating the models.

Figure 2.1 shows the usage of the EU interface when performing a simulation

Figure 2.1: Interface of the EU SESEuPy.

run with Simulink with the native interface from the command line. The simulation folder (in which the models, the modelbase and the configuration file reside) is passed as first argument to the call of SESEuPy. A second argument, which is optional and not shown in Figure 2.1, specifies whether the simulation folder shall be deleted after modeling. It can be "True" or "False". If it is "True", the simulation folder is deleted after simulation. It defaults to "False". There are two models simulated in parallel processes (see different process IDs). The execution run returns the value 0 for both processes, so the simulation was successful. Finally, the results of the simulation runs are fetched and a file with the results is created in the directory above the simulation folder. If the simulation folder is not deleted, the results can be found in CSV files in the simulation folder as well.

2.4 Building an Executable / Runnable of SESEuPy 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 SESEuPy 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 SESEuPy 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

The EU is used for automatic model generation in combination with an Experiment Control. Therefore for the EU no special examples are given.

4 Related Work

Related work with contributions to the development of the SES/MB framework is given in the documentation of SESToPy. Some related work regarding model building in connection with the SES is given in the documentation of SESMoPy.

List of Figures

List of Tables

List of Listings

Acronyms

CEA Computational Engineering und Automation.

eSES/MB extended SES/MB.

EU Execution Unit.

FMI Functional Mock-up Interface.

FMU Functional Mock-up Unit.

MB Model Base.

SES System Entity Structure.

SM simulation model.

SME Simulation Model Executable.

SMR Simulation Model Representation.