CyDER Master Algorithm

Release 1.0.0

LBNL - Building Technology and Urban Systems Division

Mar 14, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

This user manual explains how to install and use PyFMI to couple the CYMDISTToFMU with other FMUs such as a GridDyn FMU.

CHAPTER

TWO

INSTALLATION AND CONFIGURATION

This chapter describes how to install PyFMI on Windows.

Running PyFMI with Python 3.4 on Windows 32 bit

PyFMI is a python package which can be used to import and run a CYMDIST FMU. In *PyFMI* version 2.3.1, a master algorithm was added to import and link multiple FMUs for co-simulation. At time of writing, there was no *PyFMI* 2.3.1 executable available for Python 3.4 for Windows 32bit (See PyPyi.). The next steps describe requirements and steps to perform to compile *PyFMI* version 2.3.1 from source.

Note: To avoid having to recompile *PyFMI* dependent libraries from source, we recommend to use pre-compiled Windows binaries whenever available.

Requirements

The next table shows the list of Python modules and softwares used to compile version 2.3.1 of PyFMI from source so it can run with Python 3.4 on Windows 32 bit.

Install PyFMI dependencies with

```
pip install -r dev/master/bin/pyfmi-dependencies.txt
```

Below is a table with dependencies which fail to install using pip. For those, we recommend to use the MS Windows installer directly.

Modules	Version	Link
FMI Library	2.0.2 (source)	http://www.jmodelica.org/FMILibrary
Scipy	0.16.1	https://sourceforge.net/projects/scipy/files/scipy/0.16.1
lxml	3.4.4	https://pypi.python.org/pypi/lxml/3.4.4
Assimulo	2.7b1	https://pypi.python.org/pypi/Assimulo/2.7b1
PyFMI	2.3.1 (source)	https://pypi.python.org/pypi/PyFMI

Note: *PyFMI* needs a C-compiler to compile the source codes. We used the Microsoft Visual Studio 10 Professional.

Compilation

To compile *PyFMI* from source, run

```
python setup.py install --fmil-home=path_to_FMI_Library\
```

where `path_to_FMI_Library\` is the path to the FMI library.

CHAPTER

THREE

CO-SIMULATION

This section explains how to link a CYMDIST FMU with another FMU for co-simulation. In this section, we used the GridDyn FMU for the simulation coupling.

The following code snippet shows how to import and link a CYMDIST FMU (`CYMDIST.FMU`) with a GridDyn FMU (`GridDyn.fmu`).

Line 1 and 2 import the *PyFMI* modules which are needed for the coupling.

Line 8 loads the CYMDIST FMU

Line 9 loads the GridDyn FMU

Line 11 defines a vector with the CYMDIST and the GridDyn FMUs models.

Line 12 defines the connections between the CYMDIST and the GridDyn FMUs `(gridyn, "VMAG_A", cymdist, "VMAG_A")` means that the output `VMAG_A` of the GridDyn FMU is connected to the input `VMAG_A` of the CYMDIST FMU.

Line 25 passes the FMUs models and their connection to the master algorithm.

Line 27 gets the simulation option object.

Line 28 sets the communication step size.

Line 29 sets the logging to true.

Line 31 invokes the function which is used to simulate the coupled models.

```
1  from pyfmi import load_fmu
2  from pyfmi.master import Master
3  
4  def simulate_multiple_fmus():
5      """Simulate one CYMDIST FMU coupled to a GridDyn FMU.
6  
7      """
8      cymdist=load_fmu("CYMDIST.fmu", log_level=7)
9      gridyn=load_fmu("GridDyn.fmu", log_level=7)
10 
11     models = [cymdist, gridyn]
12     connections = [(gridyn, "VMAG_A", cymdist, "VMAG_A"),
13                    (gridyn, "VMAG_B", cymdist, "VMAG_B"),
14                    (gridyn, "VMAG_C", cymdist, "VMAG_C"),
15                    (gridyn, "VANG_A", cymdist, "VANG_A"),
16                    (gridyn, "VANG_B", cymdist, "VANG_B"),
17                    (gridyn, "VANG_C", cymdist, "VANG_C"),
18                    (cymdist, "KWA_800032440", gridyn, "KWA_800032440"),
19                    (cymdist, "KWB_800032440", gridyn, "KWB_800032440"),
20                    (cymdist, "KWC_800032440", gridyn, "KWC_800032440"),
```

```
                    (cymdist, "KVARA_800032440", gridyn, "KVARA_800032440"),
                    (cymdist, "KVARB_800032440", gridyn, "KVARB_800032440"),
                    (cymdist, "KVARC_800032440", gridyn, "KVARC_800032440"),]

    coupled_simulation = Master (models, connections)

    opts=coupled_simulation.simulate_options()
    opts['step_size']=1.0
    opts['logging']=True

    res=coupled_simulation.simulate(options=opts,
                            start_time=0.0,
                            final_time=1.0)

if __name__ == '__main__':
    simulate_multiple_fmus()
```

CHAPTER

FOUR

NOTATION

This chapter shows the formatting conventions used throughout the User Guide.

The command-line is an interactive session for issuing commands to the operating system. Examples include a DOS prompt on Windows, a command shell on Linux, and a Terminal window on MacOS.

The User Guide represents a command window like this:

```
# This is a comment.
> (This is the command prompt, where you enter a command)
(If shown, this is sample output in response to the command)
```

Note that your system may use a different symbol than ">" as the command prompt (for example, "$"). Furthermore, the prompt may include information such as the name of your system, or the name of the current subdirectory.

CHAPTER

FIVE

GLOSSARY

Dymola Dymola, Dynamic Modeling Laboratory, is a modeling and simulation environment for the Modelica language.

Functional Mock-up Interface The Functional Mock-up Interface (FMI) is the result of the Information Technology for European Advancement (ITEA2) project *MODELISAR*. The FMI standard is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML-files, C-header files, C-code or binaries.

Functional Mock-up Unit A simulation model or program which implements the FMI standard is called Functional Mock-up Unit (FMU). An FMU comes along with a small set of C-functions (FMI functions) whose input and return arguments are defined by the FMI standard. These C-functions can be provided in source and/or binary form. The FMI functions are called by a simulator to create one or more instances of the FMU. The functions are also used to run the FMUs, typically together with other models. An FMU may either require the importing tool to perform numerical integration (model-exchange) or be self-integrating (co-simulation). An FMU is distributed in the form of a zip-file that contains shared libraries, which contain the implementation of the FMI functions and/or source code of the FMI functions, an XML-file, also called the model description file, which contains the variable definitions as well as meta-information of the model,additional files such as tables, images or documentation that might be relevant for the model.

Modelica Modelica is a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents.

MODELISAR MODELISAR is an ITEA 2 (Information Technology for European Advancement) European project aiming to improve the design of systems and of embedded software in vehicles.

PyFMI PyFMI is a package for loading and interacting with Functional Mock-Up Units (FMUs), which are compiled dynamic models compliant with the Functional Mock-Up Interface (FMI).

Python Python is a dynamic programming language that is used in a wide variety of application domains.

CHAPTER SIX

ACKNOWLEDGMENTS

The development of this documentation was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under contract No. xxx.

The following people contributed to the development of this program:

- Thierry Stephane Nouidui, Lawrence Berkeley National Laboratory
- Jonathan Coignard, Lawrence Berkeley National Laboratory
- Michael Wetter, Lawrence Berkeley National Laboratory

CHAPTER

SEVEN

DISCLAIMERS

CHAPTER

EIGHT

COPYRIGHT AND LICENSE

Copyright

License Agreement

INDEX