Energy flexibility analysis using FMUWorld

Gusain, Digvijay; Cvetković, Miloš; Palensky, Peter

DOI
10.1109/PTC.2019.8810433

Publication date
2019

Document Version
Final published version

Published in
2019 IEEE Milan PowerTech, PowerTech 2019

Energy Flexibility Analysis using FMUWorld

Digvijay Gusain, Miloš Cvetković, Peter Palensky
Intelligent Electrical Power Grids
Faculteit Elektrotechniek, Wiskunde & Informatica (EWI), TU Delft
Delft, The Netherlands

***Abstract*—Energy flexibility is key to integrating more renewables into the grid. An essential contributor to enabling energy flexibility is P2X technologies such a Power2Heat, Power2Gas, among others. To evaluate the flexibility available from these resources and the impact they can have on the electrical grid, complex simulations need to be set up that may not always be possible using traditional simulation tools, given the multi-domain nature of such systems. Hence, the need for intelligent simulation techniques arises. This paper introduces a co-simulation tool, FMUWorld, to overcome simulation problems for complex energy systems. We use a multi-energy co-simulation and an energy flexibility analysis as use cases to explore the capabilities of the proposed tool. The ease-of-use offered by FMUWorld is shown to allow users to focus more on analysis of the system, such as parameter sensitivity, system optimisation, etc., rather than co-simulation setup. The paper highlights the key features and functionalities of FMUWorld that make it a novel tool for co-simulation of multi-energy systems.**

***Index Terms*—Cosimulation, energy flexibility, functional mock-up units, multi-energy system**

I. Introduction

In today's energy systems, modeling and simulation-based analysis are needed before anything can be implemented in the physical system. Modeling and simulation allow researchers and analysts to evaluate the impact of their decisions, identify critical operating points, and plan mitigation strategies. This process is crucial in creating a robust, reliable, and an economical energy system.

The energy system of late has been changing, and rapidly. It is transitioning from a central, unintelligent structure to a more distributed, and intelligent system. The growth in the adoption and integration of renewable energy sources (RES) such as wind, solar, etc. have contributed to this changing energy system. However, with increasing adoption of RES, challenges to keep the energy system operating reliably and securely are growing as well. Lower penetrations of RES into the grid (as is currently), do not provide any significant challenges. Once the share of RES starts becoming significant, issues such as large, and unpredictable net load ramps [1], reverse power flows [2], reduced system inertia [3], among others, start affecting reliable and economic system operation.

To tackle these problems, especially the issue of unpredictable net load ramps, the energy system has to be made flexible. This flexibility can be achieved with the tighter integration of our energy systems [4]. Coupling electric power systems with, for example, heat networks and gas networks can buffer excess RES generation, while unpredicted lack of generation can be offset by intelligent load control. To analyse the impact of energy sector coupling, a model based analysis needs to be carried out first. However, the tools used currently are domain specific, meaning they can be used to accurately model and simulate the behaviour of a single energy domain (electricity, heat, or gas).

Modeling, and simulating MES using domain-specific tools will inadvertently involve making assumptions for at least one domain. Such assumptions may limit system analysis [5]. To overcome this problem, clever software tricks to combine existing, mature, state-of-the-art tools and simulate different domains in a coupled fashion (co-simulation) can be used. There exist many co-simulation tools for MES that allow coupling of models from various energy domains, each with their advantages and disadvantages.

In this paper, we introduce a tool, FMUWorld, for quick co-simulation setup. The proposed tool enables combining models from different domains effortlessly, given the models are packaged as Functional Mock-up Units (FMUs). This allows the user to focus more on setting up analyses, such as parameter sensitivity analysis, than the co-simulation setup itself. The rest of the paper is divided as follows. Section II gives a brief overview of previous work about current state-of-art co-simulation tools and highlights the contribution of this work. Section III introduces FMUWorld, its structure, capabilities, and functionalities. Section IV provides a case study in the form of energy flexibility analysis of an industrial boiler system for food processing plants using FMUWorld. Section V discusses the results, conclusions, and future work.

II. Previous Work

Co-simulation has been a topic of active interest and research in the energy domain. There already exist quite a few co-simulation tools currently that allows users to couple models from different domains and analyse a truly smart multi-energy system. One of the tools currently available for co-simulation of energy systems is Mosaik [6]. Mosaik implements a smart grid co-simulation in an interesting fashion. It uses 4 components: SimAPI that defines communication protocol between the different simulators and Mosaik, scenario API, that allows users to create simulation scenarios, a Simulator Manager that is responsible for communicating and starting the processes of the implemented simulators, and a Scheduler that is in control of the operation of the different simulators and manages the data flow between them. Although fairly advanced, a fundamental limitation in the usage of Mosaik

978-1-5386-4722-6/19/$31.00 ©2019 IEEE

is its dependence on simulator adaptors which rely on using simulator APIs, a challenging task for non-programmers.

The dependence of co-simulation tool on simulator APIs can be avoided by packaging models as Functional Mock-up Units (FMUs). FMU of a dynamic model is a zip file that contains model equations as compiled C code and model description as an XML file. The Functional Mock-up Interface (FMI) [7] allows users to interact with the model FMU through standardised method calls. This makes it easier to exchange and couple models developed in various simulation software (provided the software provides an FMU Export capability). FMI is a growing standard, with more than 100 software tools providing the function to export their models as FMUs, including MATLAB, Dymola, OpenModelica, etc.

VirGIL is another powerful Java-based tool for hybrid energy systems co-simulation [8]. The platform allows users to connect various simulators as FMUs. However, it is not open-source software, thus limiting its availability. Also, VirGIL is not under active maintenance, which implies that it lacks newer FMI features. Mastersim [9] is another co-simulation tool that also provides a graphical user interface. Mastersim is intuitive to use, allowing users to setup co-simulation between dynamic models packaged as FMUs. The tool has only been tested for building energy performance simulations. Therefore challenges that come with an MES co-simulation are untested with Mastersim.

The motivation to develop FMUWorld is to make co-simulation an intuitive and straightforward process, which will enable new users to adopt this simulation philosophy easily and facilitate focus on designing applications, such as parameter tuning, using co-simulation. Usually, with co-simulation tools, the user tends to spend more time on ”modeling” the co-simulation setup, rather than the application of co-simulation itself. In comparison, the highly sophisticated domain-specific tools allow users easy modeling and simulation that in turn allows users to focus on analysing results obtained using these simulations. That is where FMUWorld comes in. It will be shown in this paper, FMUWorld's ease of use allows users, especially beginners, to setup co-simulation easily, while also allowing additional functionalities for advanced users.

We use two case studies to demonstrate FMUWorld's capabilities. In one case study, we set up an MES co-simulation between Closed Cycle Gas Turbine (CCGT) in an electrical network to show the straightforward process of setting up the co-simulation using FMUWorld. In the second case study, we demonstrate how easy co-simulation setup allows users to focus on setting up analyses. In this case, a parameter sensitivity analysis to analyse energy flexibility capability of an industrial boiler system.

III. FMUWorld

A. Structure

FMUWorld is based on Python 3. To interact with FMUs, two library adapters were developed. *meAdapter* and *csAdapter* to interact with Model Exchange FMUs and Co-Simulation FMUs respectively. These modules are built using the FMPy python library. The idea behind creating these modules was to make the interaction between users and individual FMUs intuitive. Using the developed adaptors, the interaction with FMUs is simplified to four simple method calls: `init()`, `step()`, `set_input()`, and `get_output()`. Apart from these, the modules also provide other advanced functions to interact with FMUs. FMUWorld is built entirely using these modules.

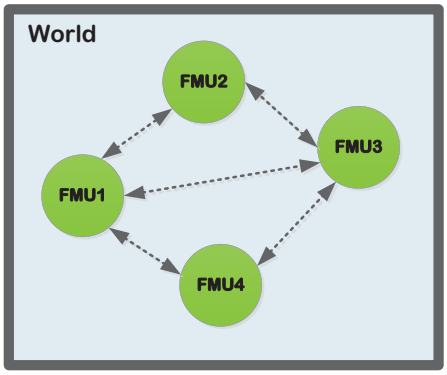

Fig. 1: FMUWorld structure

At the most basic level, co-simulations involve putting different models together and connecting their inputs and outputs for data exchange. These models may be made in any tool, using any formal modeling language. FMUWorld tries to keep this philosophy at its core. The first step in setting up co-simulation is creating a canvas/world on which, various FMUs can be embedded and connected. The canvas is known as the `World()` within the FMUWorld framework. Within `World()`, the user can define key simulation settings. These include the start time and stop time of the simulation, simulation logger, and co-simulation data exchange time interval. This is shown in Figure 1.

1) Creating World Instance and adding FMUs: `World()` can be instantiated by importing the FMUWorld module. This sets up the co-simulation master. FMUs can be added using the `add_fmu()` command. Users need to specify the fmu location, fmu instance name, the internal step size that the fmu uses, the inputs, and the outputs of the fmu. This is shown in Listing 1.

Listing 1: Importing World and adding FMUs

```
#import World object from FMUWorld
from FMUWorld import World

#create an instance of World as my_world
my_world = World(start_time = 0.0,
     stop_time = 100.0,
     logging = True,
     exchange = 1)

#add FMU to my_world
my_world.add_fmu(fmu_name = 'fmu1',
    fmu_loc = fmuLoc2,
```

```
    step_size = 1e-3,
    inputs = ['input_connector'],
    outputs = ['component.variable'])
```

It is important to explain a fundamental feature of FMUWorld here that makes it a powerful co-simulation tool for multi-energy systems. A key aspect of multi-energy systems is the timescale of different energy domains. Electrical power systems operations can range from microseconds-to-day, while thermal systems operations are evaluated in hours-to-days time scale. This means, while combining FMUs from various domains, the step sizes with which each of the FMU is designed to be simulated, will vary.

This presents a challenge of determining the instants at which information should be exchanged between the FMUs. FMUWorld handles this by analysing all the FMUs, and their step sizes defined using `add_fmu()` and determining an optimal time step size after which information should be exchanged between the FMUs. This step size for information exchange is the lowest common multiple (LCM) of all the time steps gathered from the FMUs. Figure 2 shows this approach. For a case where three FMUs have three-time steps, the FMUWorld algorithm finds a common time step which will allow individual FMUs to integrate with their time steps to reach that time, and then exchange values. In the Figure 2, the time step marked with red is the FMUWorld's calculated time step. Alternatively, if the user wants to manually input the time interval for the exchange of information between the FMUs, it can be done by changing the parameter *exchange*, defined while creating `World()` (in section III-A1).

2) Adding signals: Signals such as constant, step, pulse can be added as well. The syntax to add signals is similar to that of adding FMUs. This is shown in Listing 2.

Listing 2: Adding a signal

```
#constant signal of value x1
my_world.add_signal('signal1': [x1])
#step input. if time > t1 value = x2 else x1
my_world.add_signal('signal2': [x1, t1, x2])
#pulse input. if t1<time<t2 value = x2 else x1
my_world.add_signal('signal3': [x1, t1, t2, x2])
```

3) Defining connections between FMUs: Connections between FMUs and signals can be defined within the FMUWorld framework as dictionaries. The output-input relationships components are defined as key-value pairs of the dictionary. It is also possible to define multiple outputs to one input. Output signals are added before being set as input. This is shown in Listing 3.

4) Simulating `my_world`: Once the connections between the FMUs have been defined, the setup can be simulated. Performing simulation is achieved by calling the `simulate()` function on `my_world`. This is shown in Listing 3.

Listing 3: Adding connections and simulating

```
#define connections
connections = {'fmu1.output':'fmu2.input',
    'signal1':'fmu3.input',
    ('fmu2.output', 'fmu3.output'):'fmu4.input'}

#add connections to the world
my_world.add_connections(connections)

#simulate my_world
results = my_world.simulate()
```

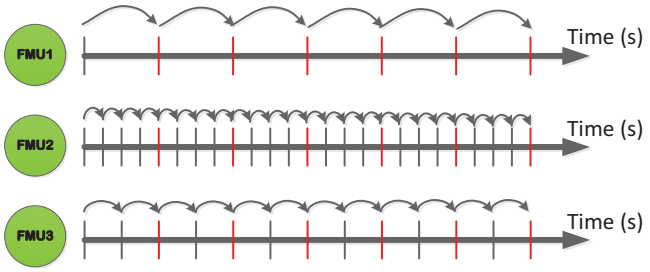

Fig. 2: Time step determination for simulation setups with variables step sizes for each FMU

If the simulation is successful, output screen print *'Success!'*. In case there was an error in the simulation run, the error is printed on the output screen. As illustrated, FMUWorld makes it extremely easy to set up and simulate FMU based co-simulation.

B. Additional Functions

In addition to setting up the simulation, advanced users can also access greater control of their simulation setup using `options()` method. FMUWorld supports the ability to make parameter changes to individual FMUs using *parameter_set*, set user-defined initial conditions to co-simulation using *init*, and modify the output signal of an FMU during the simulation before it is given to the input FMU using *modify_signal*. The signal modification allows users to scale up or scale down signals, by multiplying by a real value or adding a real value, or both. For example, turbine output in W may be needed to be input as MW into the electrical network model FMU. The functionalities are shown in Listing 4.

Listing 4: Adding options to simulation

```
#define options
my_options = {
    'parameter_set':
    {'fmu1':(['cmp.var'],[value]),
    'fmu2':(['cmp1.var1','cmp2.var2'],[val1, val2])},

    'init':
    {'fmu3':(['cmp.var'],[value])},

    'modify_signal':
    {'fmu1.output':[2], #multiplies by 2
     'fmu2.output':[1/1e6, 0.5]} #multiplies by 1/1e6
                                 #and adds 0.5
  }
```

```
#set options
my_world.options(my_options)
```

C. Result Data Structure

Once the simulation is executed as shown in Listing 3, the simulate function returns a results data object. FMUWorld uses the Pandas library to manage results; hence, the results object returned from `simulate()` is a pandas data frame.

IV. Study Case

A. Electrical-Thermal co-simulation

This case aims to evaluate the impact of the change in fuel supply to a closed cycle gas turbine (CCGT) on the system frequency using co-simulation. For this, we take two models in different energy domains: a dynamic closed cycle gas turbine (taken from [10]) for thermo-mechanical simulations, and the IEEE 9 bus system (modified from [11]) for electrical simulation. The models are tested on OpenModelica (OM) and exported as FMUs using inbuilt FMU Export function. The co-simulation setup is shown in Figure 3.

The time constant of the electrical network is 1e-3s, and that of CCGT system is 2s. Also, the output from the CCGT is in W, which needs to be converted to equivalent per unit value for the generator model in the electrical block. These two features make this setup a challenging one. The system can be set up as shown in Listing 5.

Figure 4 shows result of the co-simulation. As we can see, when the fuel input to the turbine is reduced, the mechanical power generated is reduced too. This mechanical power is transferred to the synchronous machine in the nine bus system after conversion to per unit value. As can be seen from Figure 5, the frequency of the generator also decreases.

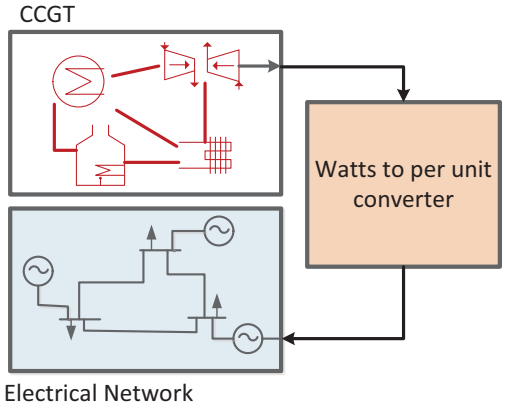

Fig. 3: Co-simulation setup combining thermal, mechanical, and electrical energy domains. The CCGT is a Model Exchange FMU, and the electrical block is a Co-Simulation FMU. The *Watts to per-unit converter* is the signal modification performed by FMUWorld.

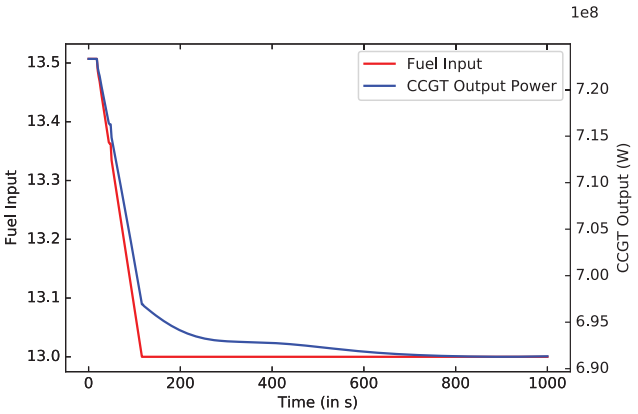

Fig. 4: Impact of reducing fuel input to the turbine. As expected, the output of the turbine decreases when the fuel input decreases.

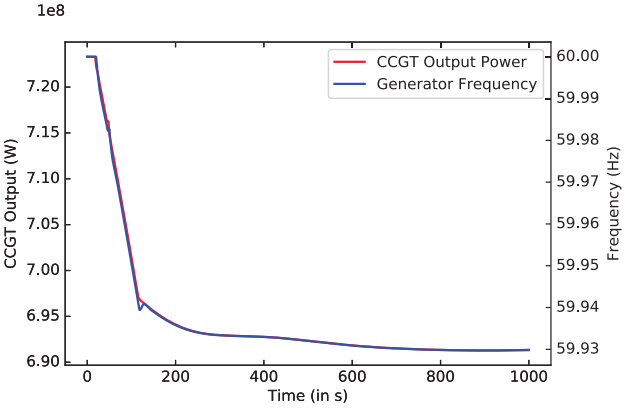

Fig. 5: Decrease in turbine mechanical power leads to decrease in frequency of the generator it is connected to.

B. Energy Flexibility Analysis

In this case, we set up a flexibility analysis for an industrial electric boiler (henceforth referred to as boiler) and storage system. Many industries, such as wood pulp, food processing, etc. require hot water for their processes. The water temperature needs to be maintained at a set value with minimum deviations. These small deviations from set value can be used to provide flexibility to the grid, especially if component capacity is large (as is the case with industrial setups).

Industrial boiler and storage systems are different from residential systems, in a sense that constraints on the fluid temperature and pressure are more strict for industrial systems. Deviations from set-point temperature cannot be sustained for long periods, and the fluid must return to set-point temperature. Given the thermal inertia of the storage tank, the temperature rise will be slow when the boiler is supplied with excess energy. Industrial components with strict constraints on

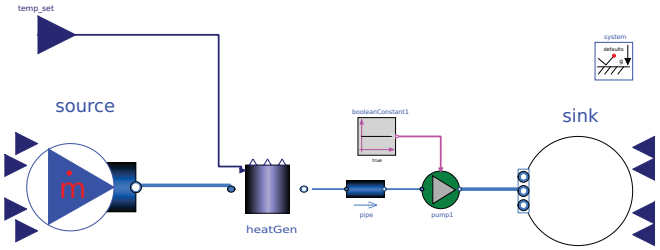

Fig. 6: Boiler and storage tank setup in Modelica. The boiler model uses the temperature set-point as an input to activate flexibility.

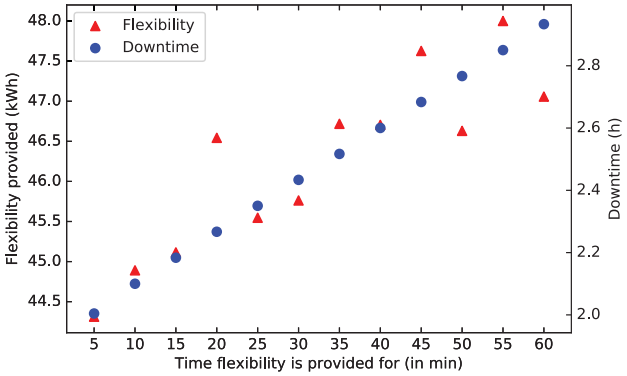

Fig. 7: Energy flexibility analysis of an industrial boiler system.

flexibility-affecting variables, such as temperature for boilers, need downtime after provision of flexibility to return to original state, during which the flexibility may not be available.

Thus, flexibility in such cases needs to be presented with a set of values: flexibility provided during activation, and the downtime following flexibility provision. To analyse the effect of various flexibility activation periods on total flexibility delivered by the boiler, and the following downtime, we set up a parameter sensitivity analysis. We run a set of simulations with various values of flexibility activation times, and evaluate total flexibility obtained and the downtime.

The idea is to show how this can be set up quickly using FMUWorld. The uncontrolled boiler-storage tank model is taken from AixLib Modelica library examples [12]. A PI controller is developed to control the amount of energy provided to the boiler. In steady state, energy is only needed to maintain the storage tank temperature by compensating for radiation losses. The boiler has a fast ramp up and ramp down rates. The boiler and storage tank model are shown in Figure 6.

The parameter sensitivity analysis can be set in FMUWorld by instantiating a simple for-loop in python. The co-simulation can be set up in a way similar to the previous test case. A boiler-storage tank FMU takes temperature set-point as input. This can be provided either by an FMU, or a signal. In our case, we provide the temperature set-point pulse input using `add_signal()`. The period of the set-point signal is changed from 5 min to 1hr in intervals of 5 minutes to determine its effect on total flexibility obtained, and the downtime. The parameter sensitivity can be set up as shown in Listing 6.

The results are shown in Figure 7. As is expected, the amount of flexibility obtained from boiler and storage unit increases when the flexibility activation time is increased. The relation between flexibility activation time and downtime is also proportional.

Similar sensitivity analysis can be done for other system parameters such as storage volume, area of the boiler, mass flow rates, boiler controller parameters, etc.

V. Conclusions

In this paper, we proposed a co-simulation tool called FMUWorld. The tool was developed to make the process of setting up multi-energy co-simulations a more natural process. We highlighted the functionalities of FMUWorld that make it a powerful tool for co-simulation, especially in MES. Key features such as multi-time step co-simulation, ability to add external signals, are effortlessly handled by FMUWorld, making it a powerful tool. It is shown through two test cases how FMUWorld enables users to focus more on setting up analyses using co-simulation, rather than co-simulations themselves.

While FMUWorld is relatively powerful for rapid setup of co-simulation, it still lacks certain features that will make it even more powerful. Some of the ideas that are still being tested include performance speed-ups by parallelizing operation. This should help significantly in large co-simulations. Another key feature is the addition of user-defined external signals. The tool is under active development and will be released as open source tool soon.

Co-simulation, especially in multi-energy systems, is a hot research topic. To get more people involved, it is needed to introduce this simulation philosophy to a newer audience, especially students. The tool is thus aimed primarily at such an audience.

Acknowledgements

This work was sponsored by Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) under the project Heat and Power Systems for Industrial Sites and Harbors (HaPSISH).

References

[1] Holttinen, Hannele, et al. ”Variability of load and net load in case of large scale distributed wind power.” 10th International Workshop on Large-scale Integration of Wind Power into Power Systems. 2011.

[2] Pollock, Jonathan, and D. Hill. ”Overcoming the issues associated with operating a distribution system in reverse power flow.” (2016): 31-6.

[3] Wang, Y., Delille, G., Bayem, H., Guillaud, X., & Francois, B. (2013). High wind power penetration in isolated power systemsAssessment of wind inertial and primary frequency responses. IEEE Transactions on Power Systems, 28(3), 2412-2420.

[4] Lund, P. D., Lindgren, J., Mikkola, J., Salpakari, J. (2015). Review of energy system flexibility measures to enable high levels of variable renewable electricity. Renewable and Sustainable Energy Reviews, 45, 785-807.
[5] Morales González, R., et al. "Optimizing electricity consumption of buildings in a microgrid through demand response." 12th IEEE PES PowerTech Conference. IEEE, 2017.
[6] Schütte, S., Scherfke, S., & Tröschel, M. (2011, October). Mosaik: A framework for modular simulation of active components in Smart Grids. In 2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS) (pp. 55-60). IEEE.
[7] Otter, M., Elmqvist, H., Blochwitz, T., Mauss, J., Junghanns, A., & Olsson, H. (2011). Functional Mockup InterfaceOverview.
[8] Rotger-Griful, S., Chatzivasileiadis, S., Jacobsen, R. H., Stewart, E. M., Domingo, J. M., & Wetter, M. (2016, June). Hardware-in-the-loop co-simulation of distribution grid for demand response. In 2016 Power Systems Computation Conference (PSCC) (pp. 1-7). IEEE.
[9] Nicolai, A., Paepcke, A. (2017, July). Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models. In Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017 (No. 132, pp. 63-72). Linköping University Electronic Press.
[10] El Hefni, B., Bouskela, D., & Lebreton, G. (2011, June). Dynamic modelling of a combined cycle power plant with ThermoSysPro. In Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany (No. 063, pp. 365-375). Linköping University Electronic Press.
[11] Baudette, M., Castro, M., Rabuzin, T., Lavenius, J., Bogodorova, T., & Vanfretti, L. (2018). OpenIPSL: Open-Instance Power System LibraryUpdate 1.5 to iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations. SoftwareX, 7, 34-36.
[12] Müller, D., Lauster, M., Constantin, A., Fuchs, M., & Remmen, P. (2016, September). AixLib-An open-source modelica library within the IEA-EBC annex 60 framework. In Proc. BauSIM (pp. 3-9).

APPENDIX

Here we present two pieces of codes used in the case studies above. The codes and the python package will soon be available on GitHub[2]

Listing 5: Test Case A

```
#import package
from FMUWorld import World
#create world instance
my_world = World(stop_time=1000,
                 logging = True,
                 exchange=2)

fmuLoc1 = os.path.join(working_dir,chp+'.fmu')
fmuLoc2 = os.path.join(working_dir,elec+'.fmu')
fmuLoc3 = os.path.join(working_dir,mod+'.fmu')

#add fmus
my_world.add_fmu(fmu_name = 'chp',
                 fmu_loc = fmuLoc1,
                 step_size = 2,
                 outputs = ['rampeQfuel.y.signal',
                            'Alternateur.Welec'])
my_world.add_fmu(fmu_name = 'elec',
                 fmu_loc = fmuLoc2,
                 step_size = 1e-3,
                 outputs = ['gen1.f',
                            'gen2.f',
                            'gen3.f'])

#define connections
connections =
    {'chp.Alternateur.Welec':'elec.machineInput'}

#add connections
my_world.add_connections(connections)

#modify signal
my_options = {'modify_signal':
              'chp.Alternateur.Welec':[1/1e6, 0.63]}
my_world.options(my_options)

#simulate
results = my_world.simulate()
```

Listing 6: Test Case B

```
#import package
from FMUWorld import World
import numpy as np, pandas as pd

#create world instance
my_world = World(stop_time=1000,
                 logging = True,
                 exchange=2)

fmuLoc1 = /location/of/boiler/storage/fmu

#dataframe for result analysis
analysis_df = pd.DataFrame()

#setup for loop for parameter sensitivity
for time in np.linspace(300, 3600, 12):

    #add fmus
    my_world.add_fmu(fmu_name = 'boiler',
                 fmu_loc = fmuLoc1,
                 step_size = 1,
                 inputs = ['pi_input']
                 outputs = ['heatGen.TCold',
                 'heatGen.THot',
                 'heatGen.heater.Q_flow'])

    #add signal
    my_world.add_signal('temp_set':
    [353.15, 5000, 5000 + time, 353.15 + 2])

    #define connections
    connections =
    {'temp_set.y':'boiler.pi_input'}

    #add connections
    my_world.add_connections(connections)

    #simulate
    results = my_world.simulate()

    #send results for processing
    flex, downtime = process_res(results, time)
    analysis_df.loc[time, 'flex'] = flex
    analysis_df.loc[time, 'downtime'] = downtime
```

[2]GitHub username: dgusain1