MOSAIK Demo Notebooks

This document outlines the usage of the MOSAIK demo notebooks provided as educational material in the context of the ERIGrid research project, state March 2019.

MOSAIK is an easy-to-use co-simulation framework that allows researchers to couple different simulation model software components to realize dynamic simulation setups of complex systems of systems. The typical application case of MOSAIK is co-simulation of cyber-physical energy systems. Each integrated simulation tool models one component of the overall system, like the distribution grid, DERs, controllers, or even energy markets. For more information on MOSAIK, see the official website http://mosaik.offis.de/. MOSAIK is an open-source tool.

While the MOSAIK framework is written in pure Python it can integrate simulators implemented via a variety of technologies. This is possible due to an architecture based on TCP sockets as well as several APIs for simulator integration supporting languages like Java, MATLAB, C#, and even the FMI standard.

Setting up an executable co-simulation setup requires the user to write a Python script following a structure defined by MOSAIK. This script is typically called a "scenario script" and it specifies which simulators are used and how they are parameterized and connected to one another. The presented set of notebooks provide executable MOSAIK scenarios that gradually become more complex. This way, users can step by step understand the handling of MOSAIK as well as the general creation of smart energy co-simulation setups for the software-based testing of new approaches.

Installation

Usage of the notebooks requires two major installations:

1. Installation of the MOSAIK framework, including the simulators for a basic demo scenario
2. Installation of Jupyter (usually a part of the Anaconda Python distribution) to actually run the notebooks

The MOSAIK demo scenario can be installed in various ways, see https://mosaik.offis.de/install/. The most basic option is a manual installation that is illustrated step by step on the website for different operating systems. Aside from that, a VM image is provided as well as an installer for certain setups. It is up to the user to decide which installation option works best for their needs. In the context of the notebooks, MOSAIK has to be started via Jupyter so that a manual installation is suggested here.

Installing Jupyter is straightforward by downloading and installing an Anaconda distribution via https://www.anaconda.com/distribution/.

In order to use all the software packages of the MOSAIK demo scenario inside Jupyter, you have to start the *virtual environment* of your MOSAIK demo in a command line. The MOSAIK installation instructions explain how to do this under different operating systems. Your command line prompt should now show the name of your virtual environment. We assume here that the virtual environment is called "mosaik", but you may have given it a different name. Next you have to execute the following commands in the virtual environment. Again, if you have chosen a different name than "mosaik", you'll have to adjust the second command accordingly.

```
(mosaik)$ pip install ipykernel
```

```
(mosaik)$ python –m ipykernel install –user –name mosaik –display-name "mosaik"
```

This will enable you to use the virtual environment with the MOSAIK software inside Jupyter. Open one of the Jupyter notebooks. Under the menu point "Kernel", you can use the option "Change kernel".

You should be able to select your MOSAIK virtual environment here (e.g. called “mosaik”). Whenever you use one of the MOSAIK notebooks, make sure that the appropriate kernel is selected for it.

Working with the notebooks

The notebooks are designed to introduce the usage of MOSAIK step by step. The first notebook “demo1” introduces a very simple power system (co-)simulation. With every following notebook, further simulation components are added to the system to make it more interesting. If you have never worked with MOSAIK before, we suggest that execute the notebooks in order and try to get a basic understanding of each one before transitioning to the next.

Demo 1

The “demo1” scenario features only two simulation components: A load flow calculation tool (PyPower) and a data source module providing domestic load data. Both tools are implemented in Python. Nevertheless, this setup will teach you already most of the basics of creating a co-simulation with MOSAIK. The scenario includes the `sim_config` structure that provides MOSAIK with the information on how to integrate a piece of simulation software.

```
sim_config = {
    'PyPower': {
        'python': 'mosaik_pypower.mosaik:PyPower'
    },
    'CSV': {
        'python': 'simulators.ws2017_csv_sim:CSV'
    }
}
```

Furthermore, the functions for starting simulators and instantiating model entities are used.

```
pypower = world.start('PyPower', step_size=15*60)
hhsim = world.start('CSV', sim_start=START, datafile=PROFILE_DATA)
```

```
grid = pypower.Grid(gridfile=GRID_FILE).children
houses = hhsim.House.create(50)
```

A data flow between the simulators is defined (Active power load data sent from the household data source to the grid nodes). Note that MOSAIK allows to connect a pool of model entities (houses) to another one (nodes) with just a single command.

```
connect_randomly(world, houses, nodes, 'P')
```

Finally, the co-simulation setup is executed.

```
world.run(until=END)
```

Aside from this, demo1 already introduces the concept of composite model entities. That means that a simulation model in the MOSAIK environment can consist of sub-models (called children). For example, a power grid model can consist of models of busses and branches. This concept is useful when integrating simulation tools into MOSAIK that model complex infrastructure like grids or communication networks. For a first time user this may seem confusing in the beginning, but a merit

is provided by the possibility to filter for model types and use the outcome to connect whole sets of entities with each other (as shown above).

```
# Filter all grid buses/nodes from the grid objects:
nodes = [element for element in grid if 'node' in element.eid]
```

The structure used here to filter for power grid nodes is a Python-specific concept called a list comprehension. Note that a normal for-loop would do the same trick. All in all, this illustrates the benefit of having a general-purpose programming language like Python for the definition of co-simulation scenarios.

Demo 2

When executing the demo1 scenario, you notice that there is no way to analyze or even look at the output of your calculations. In MOSAIK, functionalities for data storage, analysis and monitoring have to be integrated just like a simulator. Therefore, a database tool is added in demo2 to provide us with a data file at the end of the simulation to make the results accessible.

While most of the scenario remains unchanged compared to demo1, we now have an extension of the `sim_config` structure since MOSAIK needs to know how to start our database tool (that is also implemented in Python).

```
sim_config = {
    'PyPower': {
        'python': 'mosaik_pypower.mosaik:PyPower'
    },
    'CSV': {
        'python': 'simulators.ws2017_csv_sim:CSV'
    },
    'DB': {
        'cmd': 'mosaik-hdf5 %(addr)s'
    }
}
```

Just like any other simulator, the database needs to be started and instantiated.

```
db = world.start('DB', step_size=60, duration=END)
hdf5 = db.Database(filename='demo.hdf5')
```

To get the most of our simulation, we track all available output data from all our household, grid bus and grid branch entities in our database. For the household entities, this makes limited sense in this setup since they all include the same load profile. However, this functionality might be useful in the future if we ever have another domestic load model that allows parameterization of model entities.

```
connect_many_to_one(world, houses, hdf5, 'P')

busses = [elem for elem in grid if elem.type in ('RefBus', 'PQBus')]
branches = [elem for elem in grid if elem.type in ('Transformer', 'Branch')]
connect_many_to_one(world, busses, hdf5, 'P', 'Q', 'Vl', 'Vm', 'Va')
connect_many_to_one(world, branches, hdf5, 'P_from', 'Q_from', 'P_to')
```

Demo 3

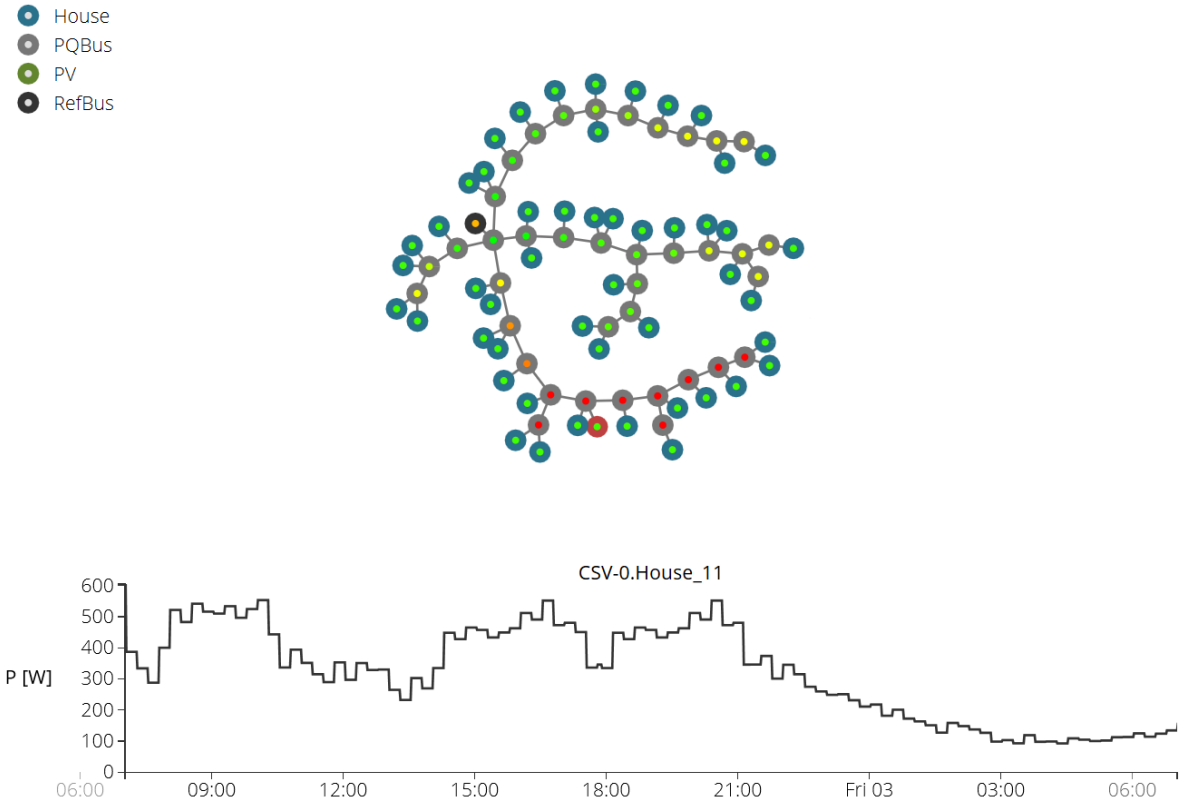
Via integration of the database module, we now get a HDF5 file with the output data from out co-simulation. We can easily load the file into an analysis setup, e.g. based on MATLAB or Python, and look at the outcome. However, for our demo scenarios, it would be nice if we had a more direct

feedback on the progress of our co-simulation. Therefore, demo3 introduces the integration of an on-line monitoring tool that allows us to visualize our simulation during runtime.

The tool we use is again Python-based on possesses some special configuration requirements, which is why we have a new line in the import section of this demo. Aside from this, all changes to the script are analogous to the ones we made for integration of the database: We extend the `sim_config` structure and add lines for initialization, instantiation and connection of the visualization tool.

```
sim_config = {
    'PyPower': {
        'python': 'mosaik_pypower.mosaik:PyPower'
    },
    'CSV': {
        'python': 'simulators.ws2017_csv_sim:CSV'
    },
    'DB': {
        'cmd': 'mosaik-hdf5 %(addr)s'
    },
    'WebVis': {
        'cmd': 'mosaik-web -s 0.0.0.0:8000 %(addr)s'
    }
}
```

The visualization tool can be employed as follows: After execution of the `world.run(…)` command (during runtime of the simulation), open a new tab in your browser and connect to the address `localhost:8000`. This should display a graph representation of your co-simulation setup. Gray circles represent power grid nodes while blue circles represent households. Colors inside a circle display severity of grid congestions (bounds are defined within the visualization tool). Clicking on a circle allows you to display the load series of that model entity over time. If the visualization should not work in your setup, you may want to try a different browser.



Demo 4

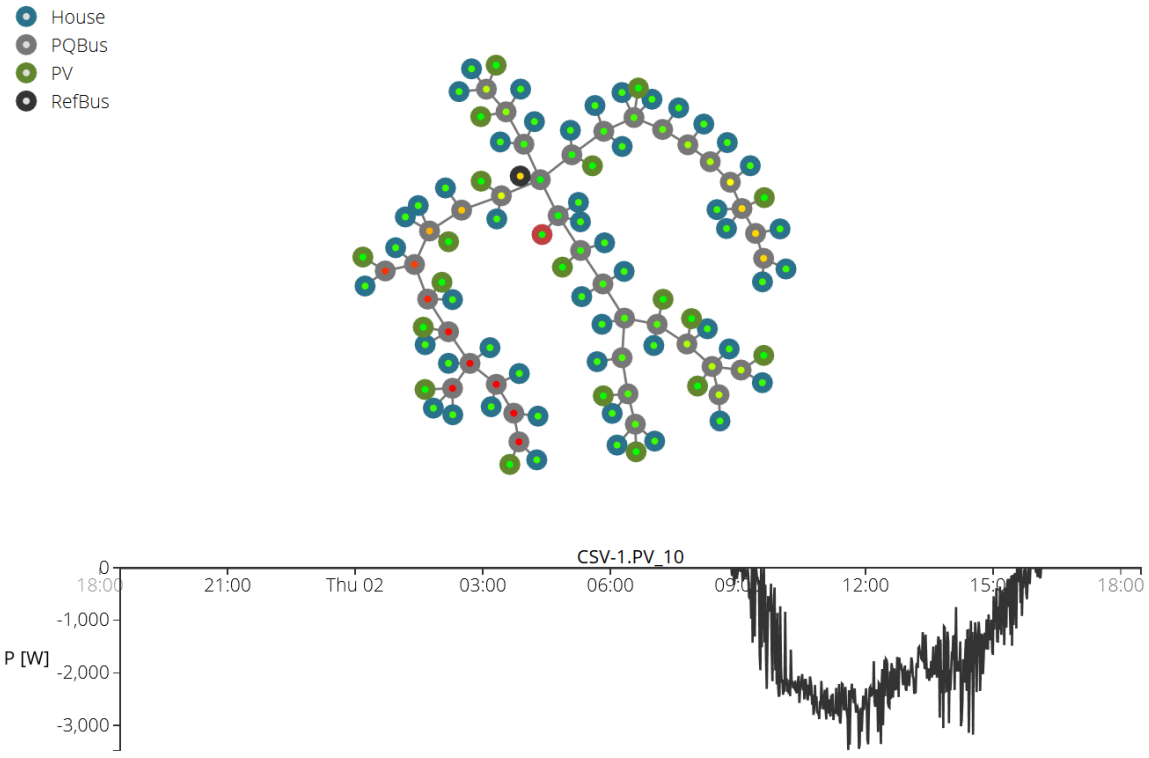
Since we now have the tools to actually monitor and analyze our co-simulation, it is time to make the overall setup more interesting by including more representations of real-world components. We start of lightly by reusing the data source tool we already have in the system to also represent PV power generation.

A new data file is now listed in our scenario parameters. And in the initialization phase we see that we can start different instances of the same simulator (if the tool allows it).

```
PV_DATA = 'data/pv_10kw.txt'
```

In the remainder of the demo script, working with the PV data source is analogous to working with the household data source. We create a set of model entities (we chose 20 here) and connect them randomly to the power grid nodes to provide generation data.

During execution, we now can see the PV units represented in our visualization tool (green circles).



Demo 5

So far we have worked with the simplest possible representation of DER in our co-simulation. We simply have included data files that provide fixed schedules over time. In the last scenario, we will integrate a more complex model to replace our data-based PV unit representation.

Again, we provide the access information about the new simulator in the extended `sim_config` structure. The underlying simulation model takes solar irradiation data as input and calculates power generation based on spherical trigonometry, considering the time of day and angle of the panel.

```
sim_config = {
    'PyPower': {
        'python': 'mosaik_pypower.mosaik:PyPower'
    },
    'CSV': {
        'python': 'simulators.ws2017_csv_sim:CSV'
    },
    'DB': {
        'cmd': 'mosaik-hdf5 %(addr)s'
    },
    'WebVis': {
        'cmd': 'mosaik-web -s 0.0.0.0:8000 %(addr)s'
    },
    'PV': {
        'python': 'simulators.ws2017_pv_simulator:PvAdapter'
    }
}
```

In the initialization phase, we now start a different type of PV simulator. Furthermore, we are using the data source tool to set up a solar irradiation data source (i.e. a “sun” for our new PV).

```
#pvsim = world.start('CSV', sim_start=START, datafile=PV_DATA)
solsim = world.start('CSV', sim_start=START, datafile=SOL_DATA)
pvsim = world.start('PV', start_date=START)
```

During model entity creation, we now see that we can use model parameters to affect the calculations in our simulation models. We are again setting up 20 PV units and specify the latitude of their placement, their size, efficiency and tilts.

```
sun = solsim.Sun()
pvs = pvsim.PV.create(20, lat=32.2, area=2.2, efficiency=0.3, el_tilt=32.2, az_tilt=0.0)
```

The connection of PV units to the grid remains unchanged since the new model, just like the old, provides active power data “P”. For the solar irradiation (direct normal irradiation, DNI), we have to connect our sun entity to all PV entities. This can be easily done via a for-loop over one-to-one connections for all PV units.

```
for i in range(len(pvs)):
    world.connect(sun, pvs[i], 'DNI')
```

All in all, we have seen in this example that it can be really easy in MOSAIK to replace a simple simulator by a more complex one. The script-based creation of co-simulation scenarios allows for things like upscaling or new connections with just a few additional lines of code or parameter changes. Therefore, MOSAIK is a suitable tool for quick prototyping of co-simulation-based test setups.

How to go on from here

The presented set of demo scenarios should have given you a first impression on the creation of co-simulation setups with MOSAIK. We encourage you to play a little with these scenarios and change some things to see what is possible with the software.

Obviously, you have to get a sense of the capabilities of your simulation tools in order to create a reasonable co-simulation setup. Therefore, the next big step for you should be to understand how simulators can be integrated into MOSAIK. You’ll find a step-by-step tutorial on this topic on the MOSAIK website (https://mosaik.readthedocs.io/en/latest/tutorials/examplesim.html). Here you will find general information about the coupling between MOSAIK and Python- as well as Java-based tools. Aside from this, you can easily couple MOSAIK with MATLAB simulation models, FMUs and just about

any tool that support TCP sockets and data in the JSON format. To learn more, we suggest that you study the MOSAIK web presence and test other example setups.