Hierarchical Multi-Level Electric Power System Simulation with Smart Photovoltaic Systems using the Functional Mock-up Interface on the Lawrencium Computing Cluster

Christoph Gehbauer[1] Joscha Müller[1]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA, {cgehbauer,joschamueller}@lbl.gov

Abstract

The adoption of distributed photovoltaics (PV) with smart inverters was one of the first large-scale deployment of grid-interactive, customer owned assets. This paper introduces a co-simulation platform for future scenarios of Distributed Energy Resources (DER), in the context of large-scale deployment, to assess the local and global impact on the electric power grid. The co-simulation platform utilizes the Functional Mock-up Interface (FMI) industry standard to couple 80,851 individual simulators. For this purpose a Modelica package named SCooDER was developed, which includes models for various DERs. The simulation was conducted at the Lawrencium high performance computing cluster. It included a hierarchical structure of multi-level electricity grids (i.e., transmission, medium-voltage distribution, and low-voltage distribution), and PV with smart inverters and time-varying load profiles at 80,000 load buses, in representation U.S. state sized electric power grid. With the flexibility of the simulation framework and the agreed-on industry standard for simulation model exchange, future applications can be very broad by coupling multi-domain simulators.

1 Introduction

Power grids world wide are undergoing big changes due to the increasing amount of distributed energy resources (DERs) such as photovoltaics (PV), behind-the-meter battery storage (BTM-BS), and electric vehicles (EVs) (Bayod-Rújula, 2009). Deploying large amounts of DER can result in unintended and potentially critical conditions on the grid. Simulations offer one means of conducting a contingency analysis to identify critical scenarios and to support the planning and installation of DERs. However, large-scale, detailed simulations are complicated to setup and solve. As a result, detailed simulations are often only conducted for a specific application due to the cost and labor intensive setup. They are also often only conducted for specific events in subsections of the grid like in (Leou et al., 2013), (Godfrey et al., 2010), or (Stetz et al., 2012). Simulations of larger grids often do not include detailed models of single DERs; instead they use pre-computed power flows (i.e., positive and negative active and reactive loads) for discrete time steps.

Another difficulty for simulations is the large variety of devices that are connected to the power grid and the resulting use cases of the simulation. In addition to the variety of DER listed above, applications typically also include buildings and commercial or industrial facilities. The challenge, therefore, is that different devices are modeled with different, domain-specific tools. These tools are often proprietary, which makes it difficult to extend or interconnect them. One solution to couple a wide variety of different domain-specific models is the Functional Mock-up Interface (FMI). Initiated by the European automotive industry, it was developed to standardize the exchange and co-simulation capabilities of models from different vendors. This standard can be used to export simulation models as a Functional Mock-up Unit (FMU), which in turn can be used in other software tools which support the FMI standard (Nouidui et al., 2019; Blochwitz et al., 2012). The source code of FMUs can be hidden, so proprietary models can be shared without revealing sensitive information.

The FMI standard is currently supported to different extents by 134 software tools (Modelica Association, s.a.). Relevant to the electric power system are the tools EMTP-RV, EcosimPro, Matlab/Simulink, and ESI SimulationX which directly support the FMI standard. Other tools such as Cymdist, PowerFactory, PSCAD, DSATools, PSS/E, Pandapower, GridDyn, MATPOWER, and OpenDSS can be supported indirectly by wrapping their Application Programming Interfaces (APIs) into FMUs using a tool called SimulatorToFMU. (Nouidui and Wetter, 2017)

In this paper we apply the FMI standard to power systems by conducting a large-scale grid simulation, which involves 80,851 FMUs in total, representing 80,000 individual customers with co-located PV systems and smart inverters connected to the grid. Smart inverters are designed to regulate the reactive power output in respect to locally observed grid voltages, to mitigate the impact of distributed PV generation on the grid. The simulation is partitioned in multiple subsections of the grid which are representative of the different voltage levels. The subsections are again encapsulated in FMUs and connected with other FMUs to form a large grid simulation. To decrease simulation time, the parallelism of subsections was applied and scaled across different compute nodes

 10.3384/ECP20169120 DOI

at the Lawrencium High Performance Computing Cluster (HPCC) at Lawrence Berkeley National Laboratory (LBNL). This paper describes the developed framework and discusses its scalability and flexibility for large-scale simulations.

2 Overview

The central part of this paper is the leverage of the FMI standard, which is used to create simulation models for a large-scale simulation of an electricity grid. The scenario includes 100 percent PV penetration (i.e., peak PV production is equal to peak load demand) and voltage-dependent smart inverter controls at each of the 80,000 individual customers. The transmission grid model is representative of a large power grid, the nominal size of a state in U.S. (i.e., Illinois).

2.1 Functional Mock-up Interface

The FMI standard describes a set of standardized functions to export and link simulation models for the use in co-simulations. A model which is exported in compliance with FMI is called an FMU. It consists of the files below which are zipped as a zip file with the ending *.fmu*:

- An Extensible Markup Language (XML) file describing parameters, inputs, outputs, and dependencies of the model.
- Compiled C-code with standardized FMI functions to evaluate the model.
- Resource data which can contain additional information such as documentation, dependency files that are required by the simulation, or graphical illustrations.

Using the FMI standard as the back-end of the simulation offers some benefits, which include (a) leveraging an agreed-upon industry standard which is well maintained, updated, and improved by industry, (b) the ability to utilize a large variety of model libraries for different domains that are built and maintained by large industry and research institutions, (c) the flexibility to couple models of different domains (e.g., battery chemistry model with EV drivetrain model coupled with the electric power grid) to form a single multi-domain co-simulation, (d) industry developed wrappers for the FMI standard in different programming languages (e.g., Python, MATLAB, Modelica, Java) that are well maintained and documented, and (e) a large community working on the FMI and related standards.

The FMI standard supports two types of simulation model export. The first mode is the co-simulation (CS) mode. In *CS* mode, every model contains its own numerical solver and provides the result for the next timestep, i.e., start time of the model plus step size, as an output. When a simulation is run in *CS* mode, the individual models are simulated with a defined timestep, and forced to advance time for the step defined. The execution time and order

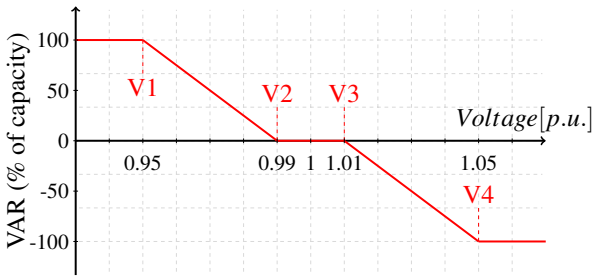

Figure 1. Volt/Var control curve for smart inverters.

is managed by an orchestrator, where one iteration completes when all FMUs are evaluated once. The other mode is the model exchange (ME) mode. *ME* models do not contain their own solver and instead provide the evaluated system of equations as output. In case of a first-order differential equation the output would be the state derivative, while *CS* would output the integrated state. An orchestrator coordinates the evaluation of models and a global solver solves the coupled models as a network. This way, the models are run repeatedly, and with a variable timestep, until inputs and outputs converge to an equilibrium (Blochwitz et al., 2012). Since *CS* can only advance in time, it is often desirable to utilize *ME* FMUs for simulation, especially when algebraic loops are present. Using *ME* FMUs, the solver can iterate back and forth in time until it finds a solution, while the *CS* FMU has to advance time. In case of a step function or other rapid change in output, the *CS* would miss the event, while *ME* allows to go back in time to find the exact event. A common workaround with *CS* FMUs are sufficiently small timesteps which, on the other hand, increase solving time. All the models in this paper are exported as FMUs and simulated by the FMI standard. The FMUs contain both types of export, depending on the hierarchy level.

2.2 Smart Inverter

To control and guide the implementation of DERs, the IEEE 1547 international interconnection standard (Basso et al., 2015) was established. In addition, some U.S. states implemented additional rules for DERs to comply with. For example, the California Public Utilities Commission (CPUC) established Rule 21 (Commission et al., 2014) for California. With respect to IEEE 1547 and Rule 21, PV inverters connected to the power grid have to provide advanced inverter functionalities. These include the capability of advanced control features like the reactive power droop control, commonly referred to as Volt/Var control. The Volt/Var control is illustrated in Figure 1.

With an active Volt/Var control, the inverter controls its reactive power generation or consumption (on the y-axis) depending on the local voltage (on the x-axis). If the local voltage exceeds a threshold, V2 or V3, the inverter starts to consume or generate reactive power in a linear response, depending on whether voltages are too low or too high. It saturates (i.e., reaches the maximal reactive power

absorption or generation) at V1 or V4 accordingly. In electrical power systems the generation of reactive power results in an increase in system voltage, and the absorption of reactive power results in a decrease in system voltage.

A Modelica package called Smart Control of DER (SCooDER) was developed by LBNL to facilitate the testing and simulation of such devices (Gehbauer et al., 2019). It contains models of PV generators, inverters, batteries, sensors, controllers, and other models relating to DER and power system simulations. The models are intended for detailed DER simulations and can all be exported as FMUs. This study used the PV generator and smart inverter control models, which are exported as FMUs using JModelica.

2.3 JModelica

JModelica is an open-source platform supporting simulation, optimization, and analysis of complex dynamic systems in the Modelica language. It provides tools for exporting and simulating FMUs (JModelica.org, s.a.). All the simulations in this paper were conducted with components of the JModelica package. PyFMI is a Python based tool for loading and executing FMUs and is part of this package. It supports simulations in either *CS* or *ME* mode by providing a simple interface to run simulations with the FMI standard in an open source environment (Andersson et al., 2016). PyFMI relies on third-part numerical solvers (i.e., CVODE as default) to solve the system of equations. CVODE is a C-based solver that can be used to solve systems of stiff and non-stiff ordinary differential equations (Cohen et al., 1996).

2.4 Pandapower

For the power grid part of the simulations, Pandapower, an open-source Python-based tool for simulating and analyzing power systems, was used. Pandapower provides power flow and optimal power flow (OPF) capabilities, which are based on PYPOWER, which in turn is based on the MATPOWER tool. Pandapower ships with a large library of pre-configured electric grid models that can be easily loaded and simulated (Thurner et al., 2018). The next section describes a tool for automated export of Pandapower as an FMU, allowing users to leverage Pandapower's power systems modeling capabilities.

2.5 SimulatorToFMU

SimulatorToFMU is a Python package developed by LBNL to wrap the high-level Python API of a third-party simulator in a Python function, which can then be exported as an FMU (Nouidui and Wetter, 2017). The utility auto-generates Modelica code that contains a model to communicate with the simulation tool through its Python API. It then invokes a Modelica translator to compile the model and export it as an FMU. It is built to leverage third-party Modelica compilers (i.e., JModelica, Dymola, or OpenModelica) to export the model. This helps to ensure the forwards and backwards compatibility with new versions of FMI. SimulatorToFMU requires an XML file, which specifies the input and the output of the simulator, as well as the Python function which interacts with the simulator. The export is implemented as a Python function call.

2.6 Lawrencium HPC Cluster

For the large-scale simulation of many FMUs, the HPCC at LBNL was used. It currently includes four clusters with a total of 924 compute nodes, ranging from 16 to 32 central processing units (CPUs) and between 64 to 128 gigabytes (GB) of random access memory (RAM) per compute node (LBNL, s.a.). It utilizes large parallel network file storage to enable fast data transfers between compute nodes. An overview of the four active stages of Lawrencium is given in Table 1.

Name	Nodes Total	Cores per Node	RAM [GB] per Node
L6	215	32	96
L5	187	20 or 28	64 or 128
L4	142	24	64
L3	380	16 or 20	64

Table 1. Active Lawrencium HPCC stages.

The simulations conducted in this study require a high number of CPU cores for efficient scale-up, and a moderate number of compute nodes. LR6 was chosen as the target cluster for this application. Note that all clusters are connected to a common private network which is utilized in this work to distribute work across multiple nodes and clusters. The framework for this work-load distribution was written in Python and exported as FMU. This FMU augments the HPCC as a single model to be used in the co-simulation.

3 Setup

This study's objective was to demonstrate the scalability and versatility of the FMI standard for power system applications. The setup was therefore focused on power grids and the impact of high penetration of PV equipped with smart inverters. What made this setup challenging was the large amount of controllable PV whose active and reactive power output impacts one another through the coupling of the electric power grid. The local grid voltage as input to the smart inverter and the reactive power as output forms a feedback loop, which is challenging to solve at large scale.

To simulate a realistic power grid, prototypical example network models of different voltage levels (i.e., 115 kilovolt (kV) transmission, 20 kV medium-voltage distribution, and 0.4 kV low-voltage distribution) were utilized to form a whole power grid. The three network models are shown in Fig. 2, Fig. 3, and Fig. 4. Fig. 2 shows the 42 bus transmission network which is based on IEEE Illinois Case 57 (Christie and Dabbagchi, 1993), Fig. 3 shows

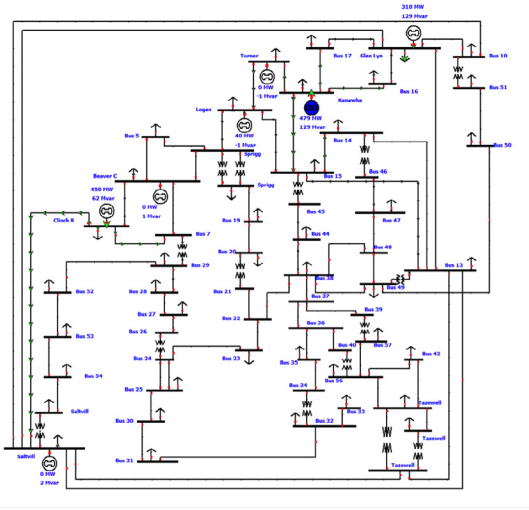

Figure 2. IEEE 57 Transmission network.

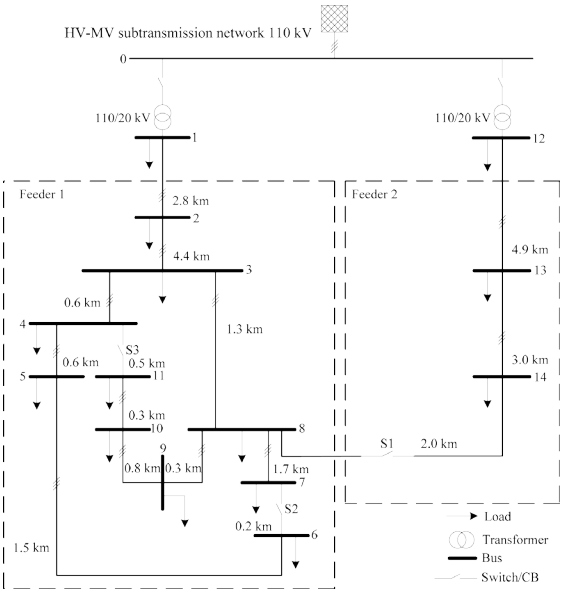

Figure 3. CIGRE Medium-Voltage network.

the medium-voltage distribution network which is based on the 13 bus CIGRE MV network (Rudion et al., 2006), and Fig. 4 shows a 146 bus low-voltage distribution network based on the Kerber LV network (Kerber, 2011). All networks are taken from the Pandapower example models. Each of the 80,000 load buses in the low-voltage distribution networks is representative of a single customer with a PV system, smart inverter, and time-varying base load. The base loads are taken from the U.S. Department of Energy (DOE) statistical reference of building types. It defines 17 load profiles for the U.S. (Department of Energy, s.a.). For this study pre-computed files for the historic weather data of San Francisco, California were used.

3.1 Time-Varying Load Profiles

The 17 DOE load profiles were clustered using the kMeans algorithm (Krishna and Murty, 1999) with the objective to establish grouping of data samples by minimizing the centroid distances. The optimal number of clusters, defined as where the loss of the algorithm is less than 20 percent, was found to be 5 clusters. In a next step one representative load profile of each cluster was manually chosen, which reduced the total number of individual load profiles to 5. This helped to reduce the overhead of assigning load profiles of similar type (e.g., *quickservicerestaurant* and *fullservicerestaurant*). The individual profiles are shown in Fig. 5 where all load profiles are scaled by its peak power demand of the selected day, which is June 1st for this study. The manually selected profiles are highlighted as a solid line, while the other profiles associated with the cluster are shown as dotted line in the same color. It can be seen that some profiles, such as the orange one which centers around the *fullservicerestaurant* profile consists of six DOE load profiles (i.e., *outpatient*, *smallhotel*, *largeoffice*, *quickservicerestaurant*, and *supermarket*) while others, such as the violet one only consist of one DOE load profile (i.e., *residential*). In either case, the grouping of DOE profiles was determined by the kMeans clustering. In order to realistically assing the profiles to load buses on the feeder model, a statistical distribution was defined, based on data of 50 individual feeder models from a California utility company.

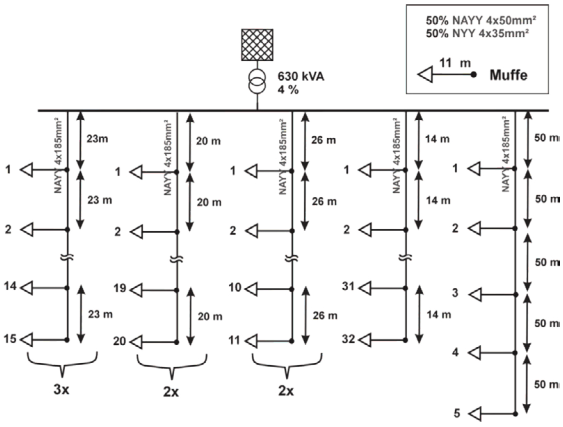

Figure 4. Kerber Low-Voltage network.

The distribution of load types is shown in Fig. 6 where the data for 50 individual feeder models is plotted as blue, orange, and green crosses for residential, industrial and commercial customers accordingly. The data is ranked by the share of residential customers. Two linear fit functions were established with the residential share as independent variable, and industrial and commercial share as dependent variables. The evaluated fit is shown as the solid line in the same color as the base data. For this study, a distribution based on the mean residential share, which is 46 percent, was chosen. The five load profiles were manually assigned to one of the three categories, and the total number of each load profile occurrence was computed. The

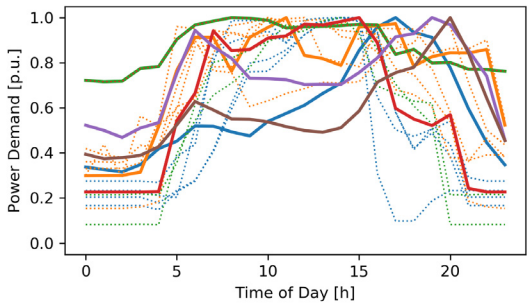

Figure 5. Overview of clustered DOE load profiles for June 1st.

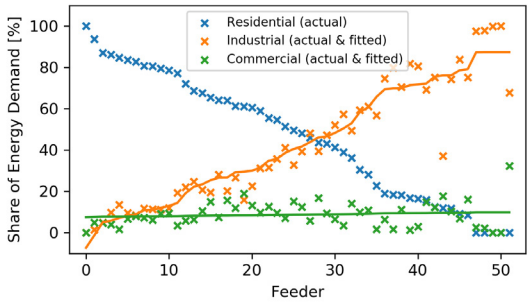

Figure 6. Load type distribution on 50 feeders of a Californian utility company.

computed mix of load profiles was randomly assigned to feeder load buses.

Since the networks are only coupled by the voltage at the interconnection point as input and the resulting active and reactive power as output, it was possible to implement scaling factors to scale single customer load demand and PV generation as aggregated power flow to the nominal of the example network model. This step avoids the over- or undersizing of customers, and results in a unique scaling factor for every load bus and feeder. This allows for a wide variety of control actuation by the smart inverter across the simulated power grid.

3.2 Functional Mock-up Units

The architecture to conduct simulations at the Lawrencium HPCC involves seven distinct FMUs which are loaded multiple times and parameterized differently, to result in a coupled system of 80,851 FMUs in total. The FMUs use both the *ME* or *CS* mode, based on the hierarchy level. The architecture is described in Section 3.3 and Fig. 8. The FMUs are described here:

- PV FMU: This FMU computes the PV generation based on historic weather data as input. It is a derivative of the PV model introduced by the Modelica Buildings Library (MBL) (Wetter et al., 2014) and modified by the SCooDER package. It uses a detailed sky model to compute incident solar irradiation on a tilted surface. The PV generation was calculated with the total irradiance with a 10 degree tilt towards south. It was exported as an *ME* FMU using JModelica. The path to the Modelica model is *SCooDER.Components.Photovoltaics.Model.PV andWeather_simple* or *Buildings.Electrical.AC. OnePhase.Sources.PVSimpleOriented* for the base version.

- Smart Inverter FMU: The function of the smart inverter FMU is to regulate the reactive power output based on locally observed grid voltages, introduced as Volt/Var control in Section 2.2. It is a linear response feedback control, based on the voltage at the interconnection point, with a deadband ($\pm$0.01 p.u.) around the control setpoint (1.0 p.u.), with saturation (0.95 and 1.05 p.u.). The maximal reactive output was set to 30 percent of the inverter size. It was implemented in the SCooDER package and exported using the *ME* API with JModelica. The path to the model is *SCooDER.Components.Controller.Model.voltVar_ param_simple_firstorder*. It also includes a first-order response to reflect the internal control delays of the smart inverter. The response time was validated with measurements taken at LBNLâĂŹs FLEXGRID facility. A positive side-effect of the control delay is the separation of algebraic loops in the co-simulation.

- Feeder Model FMU: The electrical feeder network relies on a custom Python function which calls Pandapower (a) at instantiation to create the electrical model from the example networks, and (b) on runtime to execute a power flow analysis which computes the nodal voltages based on active and reactive power flow at the load buses. The base load in the example network was scaled based on a time-varying demand profile, derived from the DOE reference buildings, as described earlier. To export the function with the SimulatorToFMU tool, another Python wrapper function was developed. To make it possible to scale-up simulations across many compute nodes at Lawrencium HPCC, the two Python functions are coupled through a socket communication, illustrated in Fig. 7. Hereby the Pandapower wrapper was encapsulated in a simple web-server and called by the SimulatorToFMU wrapper using Hypertext Transfer Protocol (HTTP) requests. Information is exchanged through JavaScript Object Notation (JSON) sanitized objects. The advantage of this implementation is the potential for large scale-up by parallelization across many compute nodes. Both Python functions were optimized for fast computation. This results in large reductions of computation time, because these functions are called at every iteration step of the feeder model. While a first implementation took up to 15 seconds to complete a single timestep, the finally implemented version completed within an average of

200 milliseconds. Further results regarding the scalability are shown in the Results section. The FMU was exported with SimulatorToFMU using the *ME* mode.

- Distribution Model FMU: The electrical distribution network also uses Pandapower to provide and solve the electric network model, and it uses the same architecture as the Feeder Model FMU. It was exported with SimulatorToFMU using the *CS* mode.

- Coupled Feeder FMU: The partitioning at Lawrencium HPCC using multiple compute nodes requires that the FMUs be executed in parallel. This is implemented in the Coupled Feeder FMU, which is a time-discrete wrapper to a coupled system of one Feeder Model FMU and PV FMU, and one Smart Inverter FMU at each of the 146 load buses. While PV generation is also present at each load bus, only one PV FMU was loaded, and the generation was scaled to 1 per unit (p.u.), which resulted in a univariate PV profile for all buses. This is a simplification made to keep the number of FMUs and connections within the coupled system low, and to allow for improved solving times. Each load bus also included the time-varying demand profile which is embedded in the Feeder Model FMU. The system of FMUs was coupled using the $CoupledFMUModelME2$ function provided by PyFMI. The Python script was wrapped using the described web-server approach and once-again exported using the *CS* API with SimulatorToFMU. While the exported Coupled Feeder FMU appears to raise state events, the actual implementation of the underlying C function does not support this functionality. However, the wrapped coupled feeder system does support the indication and handling of state events (e.g., when control loops saturate). To simplify this assumption, the Coupled Feeder FMU can be seen as discrete, while the embedded coupled feeder system is continuous with state event handling by the CVODE solver of PyFMI. The tolerances for CVODE were modified as 1e-3 for $atol$ to meet the accuracy of Pandapower.

- Transmission Model FMU: This FMU encapsulates the transmission model in Pandapwer. It is similarly structured to the Distribution Model FMU and also exported as FMU using SimulatorToFMU with the *CS* mode.

- Coupled Distribution FMU: The highest level of aggregation is the Coupled Distribution FMU, which encapsulates one Distribution Model FMU, 13 Coupled Feeder FMUs, and one custom orchestrator. It was exported using SimulatorToFMU with the *CS* mode.

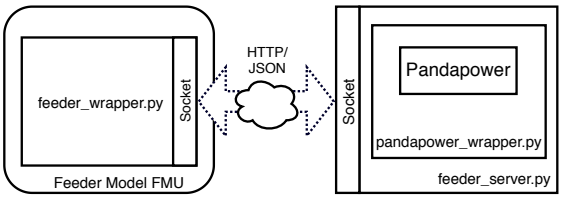

Figure 7. Illustration of the socket-based communication between the FMU wrapper and the simulator. The simulator can be hosted locally or remotely across the Lawrencium HPCC.

3.3 Architecture

The architecture for computation reflects the electrical network by decoupling the system at the different voltage levels. As described previously, each load bus of the transmission model is connected to one coupled distribution network. Each distribution model is connected to 13 coupled feeder networks, which are then in turn connected to the PV, smart inverter, and time-varying load at each feeder load bus. The architecture is illustrated in Fig. 8.

The architecture combines the Feeder Model FMU, PV FMU, and Smart Inverter FMU into a coupled system of FMUs, all exported using the *ME* mode. This system is solved for a specific time step using the $CoupledFMUModelME2$ function provided by PyFMI, and is again wrapped as an FMU to form the Coupled Feeder FMU. The Coupled Feeder FMU reflects a fully populated feeder where state events of models (e.g., deadband or saturation of the smart inverter control) are considered and solved by PyFMI. The timestep within the Coupled Feeder FMU is variable based on the solver of PyFMI. One level higher on the medium voltage distribution level, the Distribution Model FMU is coupled with the Coupled Feeder FMU by a custom orchestrator which employs the *CS* mode of the FMUs. These FMUs are discrete in time and are invoked based on the high-level time step. This system of FMUs was again wrapped as another FMU, namely Coupled Distribution FMU. It was loaded for each load bus of the Transmission Model FMU and execution was handled by a custom orchestrator that utilizes PyFMI.

The partitioning of the FMUs on the Lawrencium HPCC was determined by a total of 546 Coupled Feeder FMUs, which each were solved in parallel. It was most practical to assign five Coupled Feeder FMUs to one compute node at HPCC. The LR6 Lawrencium HPC cluster, which provides 32 logical CPU cores per compute node, was used for this study. The Transmission Model FMU consisted of 42 load buses, each connected to one Coupled Distribution FMU. This translates to a total of 12 LR6 compute nodes for the simulation. With 215 compute nodes available at LR6, this would scale to a maximum of 1,075 Coupled Distribution FMUs. With 1,898 load buses per Coupled Distribution FMU, the maximal number of individual customers would be about 2 million. However, solving clock-time could be traded against scaleup to fur-

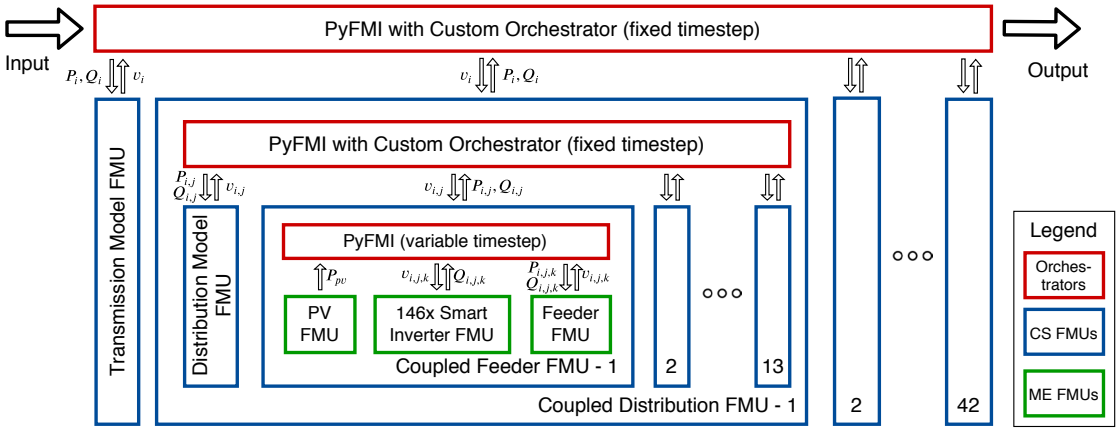

Figure 8. Hierarchical multi-level simulation architecture with high-level inputs and outputs for each FMU. The exchange variables are active power, P, reactive power, Q, and voltage, v. The three hierarchical voltage layers are transmission, denoted by i with 42 load buses, medium-voltage distribution, denoted by j with 13 load buses, and low-voltage distribution, denoted by k with 146 load buses.

ther increase the number of individual loads per node.

3.4 Orchestrator

The co-simulation of multiple independent models requires the models to be in a standardized format, in this case the exported FMUs, and an orchestrator to coordinate the execution of FMUs and to solve the system of equations. The Coupled Feeder FMU and Coupled Distribution FMU are special cases because they wrap a whole co-simulation to form a new FMU. In case of the Coupled Feeder FMU it already includes an orchestrator which in this case is PyFMI. However the top-level coordination of the Coupled Distribution FMU and transmission level is implemented with a custom algorithm which controls (a) the primary timestep of the model, and (b) the convergence of the model. While the timestep coordination is implemented as a simple *for* loop, the convergence is more involved by requiring an iteration algorithm and convergence objective and criteria. The implemented algorithm is limited to a maximal number of 10 iterations, with a convergence criteria of voltage derivative less than 5e-3 per unit. The iteration sequence starts with the initialization of the grid model FMU whose computed bus voltages represent the inputs to the coupled FMU. In order to support the solver in its action, the voltage derivatives are dampened by a first-order delay, until an equilibrium is reached. This delay dampens the control action of the decentralized control loops of the underlying Smart Inverter FMUs, to avoid oscillatory behavior. Note that this delay does not affect the physical behavior of the system, and is solely implemented as part of the solving algorithm. The coupled FMU is evaluated in parallel, and the resulting active and reactive power at the feeder heads are fed directly into a last evaluation of the grid model FMU. The bus voltage derivatives are computed, and the system is checked for convergence. If it did not converge and if it also did not reach the maximal number of iterations, another iteration is initiated. Otherwise the iteration is terminated and the result returned.

One drawback with coupling the FMUs exported with the *CS* mode is the timestep control. *CS* FMUs can only advance time which is problematic when seeking the convergence of a system at a defined simulation time. The workaround are sufficiently small timesteps where dynamics within the FMU are close to constant. In this case the internal timestep for an iteration was set to 10 milliseconds.

4 Results

The co-simulation was conducted for one sunny day with an hourly timestep on the transmission level. As described earlier, the low-level Coupled Feeder FMU, which encapsulates the PV generation, time-varying base load, and smart inverter, was solved with a variable timestep. The Fig. 9 provides an overview of the smart inverter actuation and solving time for all of the 80,000 individual customers.

The first subplot shows the statistics of the time-varying base load as boxplot normalized to each customers nominal active power demand, for each hour of the simulation. The statistics are based on each of the 80,000 individual customers. The secondary axis of the first subplot, in red, shows the univariate PV profile which is applied to all customers, also normalized to the customer size. The second subplot shows the distribution of smart inverter control actuation as reactive power scaled to the nominal reactive power of each customer. The third subplot shows the distribution of feeder head voltage, as determined by the transmission and medium-voltage distribution network. It is scaled to the nominal feeder voltage. The fourth subplot shows statistics of the solving time, as boxplots on the primary axis, and maximal number of iterations, in red on the secondary axis. Both statistics are based on the 546 Coupled Feeder FMUs. The mean simulation time

 10.3384/ECP20169120 DOI

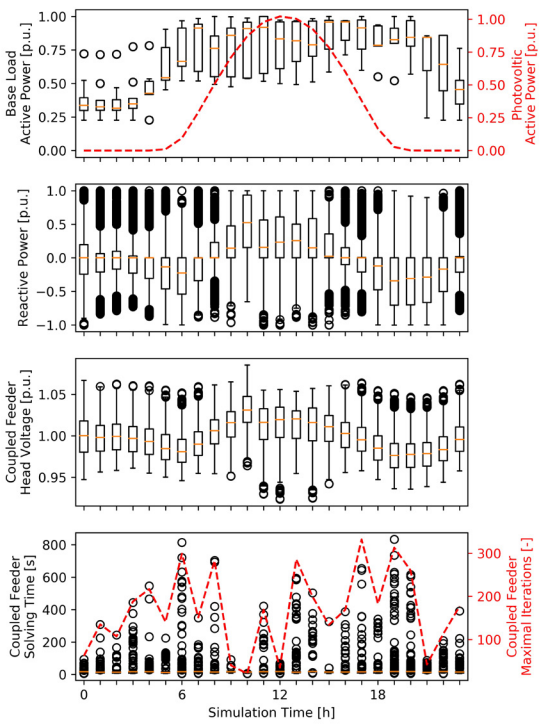

Figure 9. Overview of full-scale grid results with 80,000 individual customers.

per iteration is 25 seconds across all timesteps, whereas the standard deviation ranges between 25 to 175 seconds, depending on the timestep.

5 Discussion and Future Work

This paper successfully demonstrated the distributed co-simulation of various components of an electric power grid. This includes time-varying load and PV generation, feedback control loop of a smart inverter with Volt/Var control, and electrical coupling through multiple voltage levels.

While the setup simplified the complexity of electric distribution networks, customer load variation, and site-dependent PV generation, it demonstrated the capabilities of the developed SimulatorToFMU tool and the FMI overall to solve complex large-scale cross-platform co-simulation setups. The setup included individually validated models from publicly available Modelica libraries, in case of the Smart Inverter FMU and PV FMU, or Pandapower package, in case of the grid models. The coupled system of FMUs was solved using the PyFMI package which was developed and validated by its developer Modelon. While the whole system results could not compared to other simulators, mainly due to lack of availability to couple those individual models, which was the motivation of this paper, the project team believes that the performed validations are sufficient.

The introduced simulation setup exploited the potential of reduced complexity for high-level aggregated models, while keeping a full detail on the lower-level models. An aggregation of 80,000 individual customers was used to represent the interaction on a U.S. state sized electric power grid. The level of scale, i.e., about 50 times scaleup in this example, was achieved by reducing the number of feeders per substation to one, as well as the number of medium-voltage distribution circuits per transmission node to one. One example where the low-level detail of individual customers is important would an over-voltage event to trip smart inverters which in turn would suddenly reduce the voltage, and could end-up with a cascading effect triggering neighboring substations or feeders. In particular the setup of a coupled co-simulation system being wrapped as an FMU is important to capture such state events of smart inverters. The aim of this wrapped system was to simplify the algorithm and to facilitate a effective separation of low-level FMUs (i.e., PV FMU and Smart Inverter FMU), mid-level FMUs (i.e., Distribution Model FMU and Coupled Feeder FMU), and high-level FMUs (i.e., Transmission Model FMU and Coupled Distribution FMU) with the objective of reducing solving time. The complexity with this setup was the need to evaluate the coupled systems in parallel, while the coupling on higher levels required a serial evaluation. In addition the partitioning on compute nodes on the Lawrencium HPCC required the discrete separation of the models on compute nodes. One drawback of the current implementation is the limitation that state events cannot be propagated to higher-level FMUs. In case of the cascade effect described earlier, a higher-level distribution or transmission system would not recognize the event until the next timestep was reached. This limitation originates in the implementation of the developed SimulatorToFMU tool to export Python scripts as FMUs. As described in Section 3.2, the Coupled Feeder FMU appears to raise state events, but the actual implementation of the underlaying C function does not support this functionality. This is a recognized limitation of SimulatorToFMU, and it is being discussed for future implementation by providing a zero-crossing function to project and raise such state events.

Another simplification taken in this paper was the export of the high-level FMUs with the *CS* API, while lower-level FMUs use the *ME* API. As described in the Introduction, *ME* allows to extract derivatives and roll-back time, which are important features to solve coupled systems. The *ME* model is therefore generally preferred for the application in power systems, especially when algebraic loops (e.g., from feedback controllers) are present. However *ME* requires an external solver and handling of events, whereas *CS* internally solves this problem. It is therefore easier to interface with *CS* FMUs, to build a customized orchestrator. In addition, the added functionality of state events would not be exploited because of the previously discussed limitation of SimulatorToFMU.

For future work, simulations with real load profiles could be done. Due to the lack of open source load pro-

files based on real measurements, this study used the simulated OpenEI load profiles for DOE reference buildings. This results in less variety of loads and therefore less realistic scenarios. Nevertheless, realistic and authentic data are not necessary to proof the functionality of the large scale simulation itself. The connection between all levels of FMUs was proven in this paper. For more detailed real world scenarios simulated with the developed platform, a higher variety of real load profiles would be preferable.

In future it also might be possible to upgrade the architecture with the FMI 3.0 standard (alpha version released). The proposed functionalities include ports and icons, array variables, clocks and hybrid co-simulation, binary data type, intermediate output values, and source code FMUs (FMI 3.0, 2018). Especially the clocks and hybrid co-simulation functionality, which allows the triggering of events in co-simulation mode, can have promising applications in large scale simulations. Longer timesteps can be used, without missing events in single FMUs and the resulting effects those effects would have on other FMUs. Because of the industry based nature of the FMI standard and the growing support of it, further developments to improve the usability and capabilities of the FMI can be expected.

6 Conclusion

The results of this paper demonstrate that the FMI standard offers a promising way to create large scale simulations. It is an easy and convenient way to combine a wide variety of models on a large scale, but to still simulate them with a high level of detail. The developed SimulatorToFMU tool enables users to further increase the number of available model libraries and simulation tools spanning into the power systems domain. While this paper focused on PV systems with smart inverters, many future scenarios involving multi-domain models could be conducted. Examples are detailed building simulations to replace simple load profiles, electric vehicles with individual availability and constraints, or advanced control systems such as Model Predictive Control for building heating, ventilation, and air conditioning systems to assess load shifting capabilities on a large scale.

Simulating large parts of the power grid with multi-level and high-fidelity resources provide a more realistic result compared to plain simulations of single voltage level and aggregated generation. By connecting many distribution feeders with a transmission grid model, the impact of feeders on one another is taken into account, which helps to reveal global grid challenges such as steep ramping demand in a high-PV deployment scenario.

This paper also demonstrated the ability of FMI and other tools to scaleup to about eighty thousand individual FMUs evaluated in a co-simulation. The multi-level hierarchical partitioning of the simulation into many parallel models, which run independently on different compute nodes at the Lawrencium HPCC, greatly reduces the simulation time and allows for a fidelity which is often not achievable on a single computer.

Acknowledgment

This work was supported by the U.S. Department of Energy SunShot National Laboratory Multiyear Partnership (SuNLaMP) program award number 31266 and the California Energy Commission (CEC) Electric Program Investment Charge (EPIC) solicitation PON-14-303.

References

Christian Andersson, Johan Åkesson, and Claus Führer. *Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface*. Centre for Mathematical Sciences, Lund University, 2016.

Thomas Basso, Sudipta Chakraborty, Andy Hoke, and Michael Coddington. IEEE 1547 Standards advancing grid modernization. In *2015 IEEE 42nd Photovoltaic Specialist Conference (PVSC)*, pages 1–5. IEEE, 2015.

Angel A Bayod-Rújula. Future development of the electricity systems with distributed generation. *Energy*, 34(3):377–383, 2009.

Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 173–184. Linköping University Electronic Press, 2012.

Rich Christie and Iraj Dabbagchi. IEEE 57-bus system. 1993.

Scott D Cohen, Alan C Hindmarsh, and Paul F Dubois. CVODE, a stiff/nonstiff ODE solver in C. *Computers in physics*, 10(2): 138–143, 1996.

California Energy Commission et al. Rule 21 Smart Inverter Working Group Technical Reference Materials. *California Energy Commission*, 2014.

Department of Energy, s.a. Commercial and Residential Hourly Load Profiles for all TMY3 Locations in the United States, s.a. `https://openei.org/doe-opendata/dataset/commercial-and-residential-hourly-load-profiles-for-all-tmy3-locations-in-the-united-states`, Accessed: 2019-09-24.

FMI 3.0, 2018. FMI 3.0-alpha feature list. `https://fmi-standard.org/news/2018/05/30/fmi-3-0-alpha-feature-list.html`, 2018. Accessed: 2019-10-27.

Christoph Gehbauer, Joscha Mueller, Tucker Swenson, and Evangelos Vrettos. Photovoltaic and Behind-the-Meter Battery Storage: Advanced Smart Inverter Controls and Field Demonstration. `https://github.com/LBNL-ETA/SCooDER`, 2019.

Tim Godfrey, Sara Mullen, David W Griffith, Nada Golmie, Roger C Dugan, and Craig Rodine. Modeling smart grid applications with co-simulation. In *2010 first IEEE International conference on smart grid communications*, pages 291–296. IEEE, 2010.

JModelica.org, s.a. JModelica.org. `https://jmodelica.org`, s.a. Accessed: 2019-09-26.

Georg Kerber. *Aufnahmefähigkeit von Niederspannungsverteilnetzen für die Einspeisung aus Photovoltaikkleinanlagen*. PhD thesis, Technische Universität München, 2011.

K Krishna and Narasimha M Murty. Genetic K-means algorithm. *IEEE Transactions on Systems Man And Cybernetics-Part B: Cybernetics*, 29(3):433–439, 1999.

LBNL, s.a. High Performance Computing at Berkeley Lab. `https://sites.google.com/a/lbl.gov/lrc/home`, s.a. Accessed: 2019-09-27.

Rong-Ceng Leou, Chun-Lien Su, and Chan-Nan Lu. Stochastic analyses of electric vehicle charging impacts on distribution network. *IEEE Transactions on Power Systems*, 29(3):1055–1063, 2013.

Modelica Association, s.a. List of tools supported by FMI. `https://fmi-standard.org/tools/`, s.a. Accessed: 2019-10-27.

Thierry Nouidui and Michael Wetter. SimulatorToFMU v0. 1. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2017.

Thierry Nouidui, Jonathan Coignard, Christoph Gehbauer, Michael Wetter, Jhi-Young Joo, and Evangelos Vrettos. Cyder–an fmi-based co-simulation platform for distributed energy resources. *Journal of Building Performance Simulation*, 12(5):566–579, 2019.

Krzysztof Rudion, Antje Orths, Zbigniew A Styczynski, and Kai Strunz. Design of benchmark of medium voltage distribution network for investigation of DG integration. In *2006 IEEE Power Engineering Society General Meeting*, pages 6–pp. IEEE, 2006.

Thomas Stetz, Frank Marten, and Martin Braun. Improved low voltage grid-integration of photovoltaic systems in Germany. *IEEE Transactions on sustainable energy*, 4(2):534–542, 2012.

Leon Thurner, Alexander Scheidler, Florian Schäfer, Jan-Hendrik Menke, Julian Dollichon, Friederike Meier, Steffen Meinecke, and Martin Braun. Pandapower–An open-source python tool for convenient modeling, analysis, and optimization of electric power systems. *IEEE Transactions on Power Systems*, 33(6):6510–6521, 2018.

Michael Wetter, Wangda Zuo, Thierry S Nouidui, and Xiufeng Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014.