Enabling Smart Grid Cosimulation Studies

By Timothy M. Hansen, Rahul Kadavil, Bryan Palmintier, Siddharth Suryanarayanan, Anthony A. Maciejewski, Howard Jay Siegel, Edwin K.P. Chong, and Elaine Hale

Rapid design and development of the technologies and controls.

IMAGE LICENSED BY INGRAM PUBLISHING

THE 21ST CENTURY ELECTRIC POWER GRID is transforming with an unprecedented increase in demand and increase in new technologies. In the United States Energy Independence and Security Act of 2007, Title XIII sets the tenets for modernizing the electricity grid through what is known as the "Smart Grid Initiative." This initiative calls for increased design, deployment, and integration of distributed energy resources, smart technologies and appliances, and advanced storage devices. The deployment of these new technologies requires rethinking and re-engineering the traditional boundaries between different electric power system domains (Figure 1).

Historically, the analysis and operation of electric power systems has used isolated subdomain tools for the transmission grid and wholesale markets, the distribution grid, and end users. Each tool models one subdomain well, but makes engineered simplifications of the other subdomains with which it interacts. As more smart grid technologies [e.g., distributed photovoltaics (PVs), electric vehicles, and solid-state transformers] are integrated into distribution grids, static distribution simulations and

Digital Object Identifier 10.1109/MELE.2015.2509899
Date of publication: 1 March 2016

2325-5987/16©2016IEEE

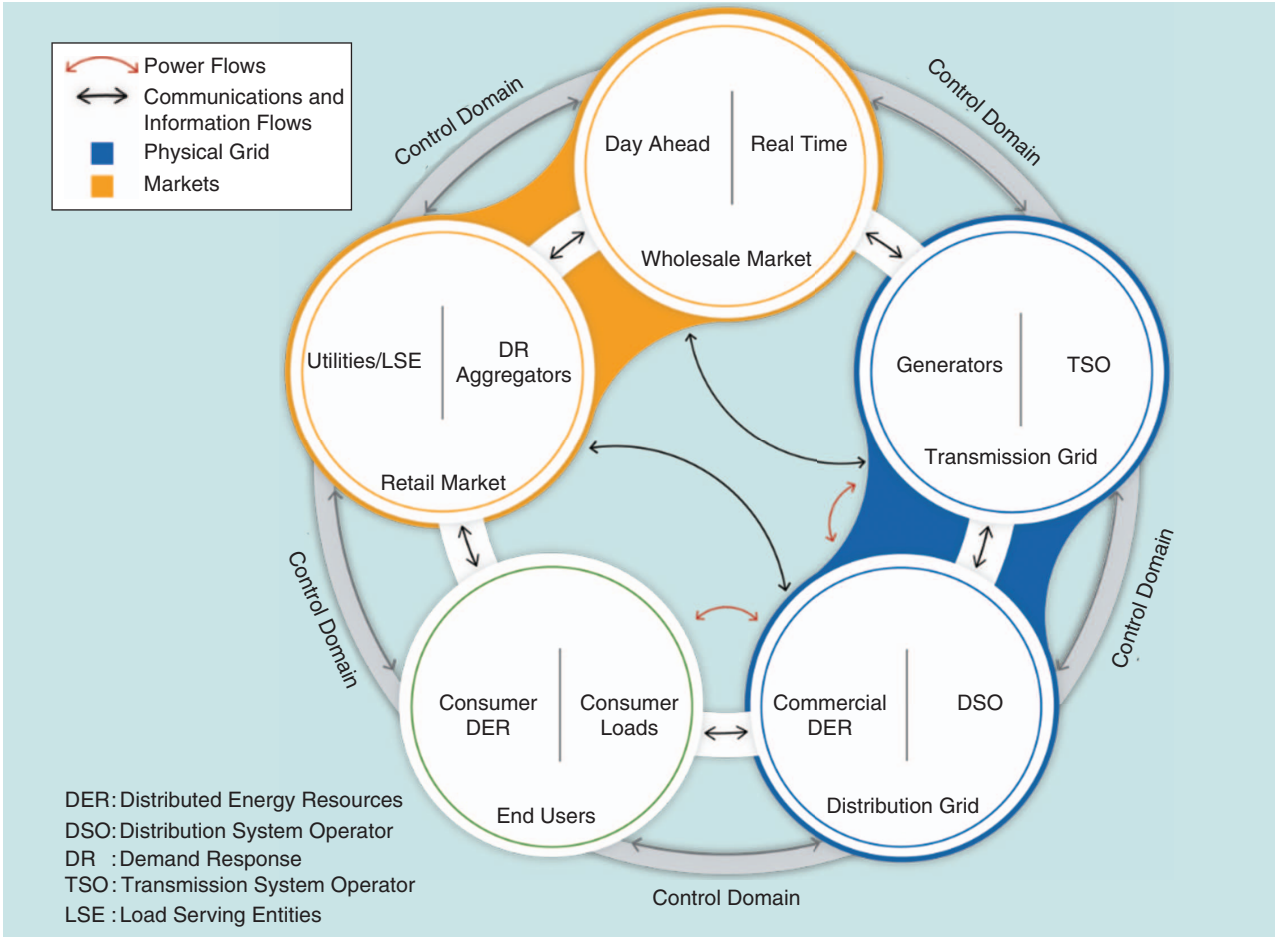

Figure 1. Individual domains in the electric power system and their interactions. (Figure courtesy of Julieta Giraldez, NREL.)

simple large-area time-series models of loads in transmission simulations are no longer sufficient for performing modeling and analysis. The integration of these new technologies and their control algorithms into the existing grid require complex control methods and extensive testing, and, at large enough deployment, they will have an impact outside of the distribution system, with visibility into the transmission grid and wholesale market. By using tools that only model one domain, the complex interactions between the new smart grid distribution system and the bulk power network are lost, and their impact on the grid may be understated. This necessitates the use of cosimulation tools—multiple tools, each modeling a single domain in detail, that interact while running simultaneously.

The future electric grid is emerging at the intersection of the physical networks, electric markets, and the end user. To enable the cosimulation of the transmission grid, distribution grid, wholesale and retail markets, and end users, we have been extensively developing open source open source tools in a combined effort between South Dakota State University, Colorado State University, and the U.S. National Renewable Energy Laboratory (NREL). Specifically, we have been developing tools to model and control distributed energy resources in GridLAB-D, an agent-based distribution system simulator, coordinated at the transmission system level, including ac power flow and wholesale markets. The three tools presented in this article are

- the Data Exchange Model (DEx.py)—a combined top-down and bottom-up end-user load profile and power network model creation tool
- Bus.py—a communication interface for communicating with GridLAB-D and controlling distributed energy resources during GridLAB-D's execution (links to the open-source code for the tools can be found at http://www.engr.colostate.edu/sgra/)
- the Integrated Grid Modeling System (IGMS)—a high-performance computing (HPC)-enabled independent system operator (ISO) to appliance-scale electric power system modeling cosimulation platform.

The tools exist at two stages in the cosimulation process: 1) network creation and population and 2) during simulation run time. In Stage 1, we generate the physical power networks at the transmission level for use in MATPOWER and at the distribution level for use in GridLAB-D. As a part of this process, we populate the distribution-level feeders with extensive end-user assets and load profiles for large-scale distribution-level simulations using DEx.py.

We developed a communication framework for facilitating the cosimulation during run time in Stage 2). The framework,

called Bus.py, is a software transmission bus interface for use in smart grid cosimulation studies. An abstract interface is provided in Python that includes interprocess communication to the distribution simulator GridLAB-D. The principle of Bus.py is a flexible, easy-to-use interface to enable the cosimulation of electric power system tools. As use cases, Bus.py enables bidirectional power flow studies resulting from increased distributed energy resources (Figure 2) and allows the study of the physical responses on distribution feeders and wholesale and retail markets from a multitude of customer home energy management systems (HEMSs) and other smart grid technologies (Figure 3). Each tool is described in detail, with case studies demonstrating the usefulness of cosimulation. By enabling a variety of scenarios that cross traditional electric power system domain boundaries, cosimulation will enable the rapid design and development of the technologies required for the future power grid.

End-User and Power Network Modeling Using DEx.py

In the emerging smart grid, many new technologies are being introduced at the distribution-level. These new end-user assets, such as distributed PV solar, smart appliances, and smart heating, ventilation, and air-conditioning systems, may impact the transmission grid and wholesale market level unlike in the past. To properly determine the impact of these technologies and their controls on power system network and market operations, we need methods for populating distribution feeders with differing amounts of these technologies stochastically while still maintaining a realistic aggregate load curve at the point of common coupling (PCC) with the transmission level. To that end, we developed the DEx.py in the programming language Python (Figure 4).

DEx.py is a platform for creating a combined transmission and distribution network model, each in formats that are used by MATPOWER and GridLAB-D, respectively, given a customizable set of network input parameters. DEx.py consists of two high-level steps: network creation and end-user load population. To facilitate ease-of-use, DEx.py presents a graphical user interface (Figure 5) for the creation of complex distribution topologies that form an input to GridLAB-D, each existing at a transmission bus interconnected by a transmission system modeled in MATPOWER. As an example, consider the Roy Billinton Test System shown in Figure 6. At each load bus, a distribution system modeled in GridLAB-D with a matching peak and realistic end-user models must be generated using DEx.py.

Depending on the peak load at the PCC and the settings of the types of distribution feeders and transformers, DEx.py will generate a GridLAB-D distribution feeder with matching residential loads connected to a set of load points (Figures 7 and 8). Each transmission or subtransmission node can be expanded into a hierarchal distribution network of up to four levels. For each transmission node, the distribution hierarchy consists of n feeders (level 1). Each primary feeder will

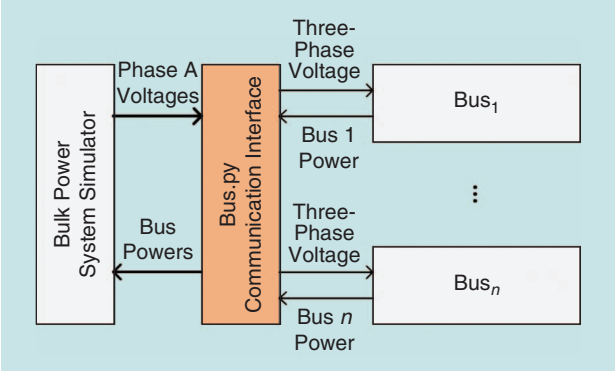

Figure 2. Bus.py interfacing with a bulk power simulator (MATPOWER) and many individual transmission buses, each being modeled in GridLAB-D.

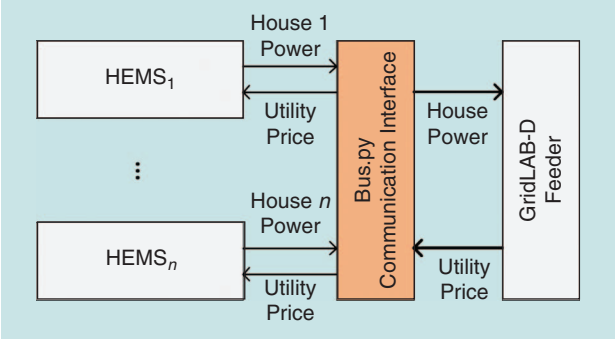

Figure 3. Bus.py interfacing a GridLAB-D simulator with many HEMSs.

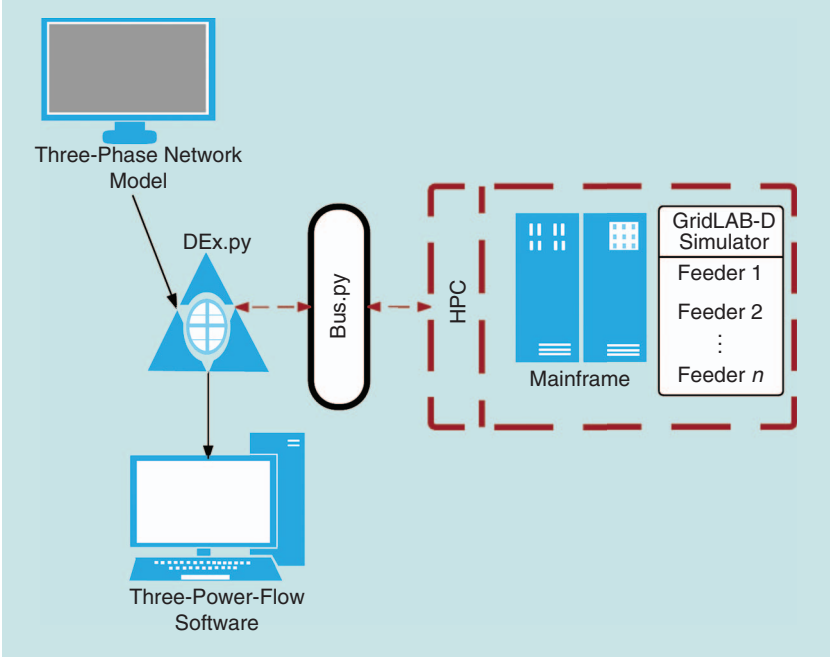

Figure 4. DEx.py as a presimulation network and end-user model generator.

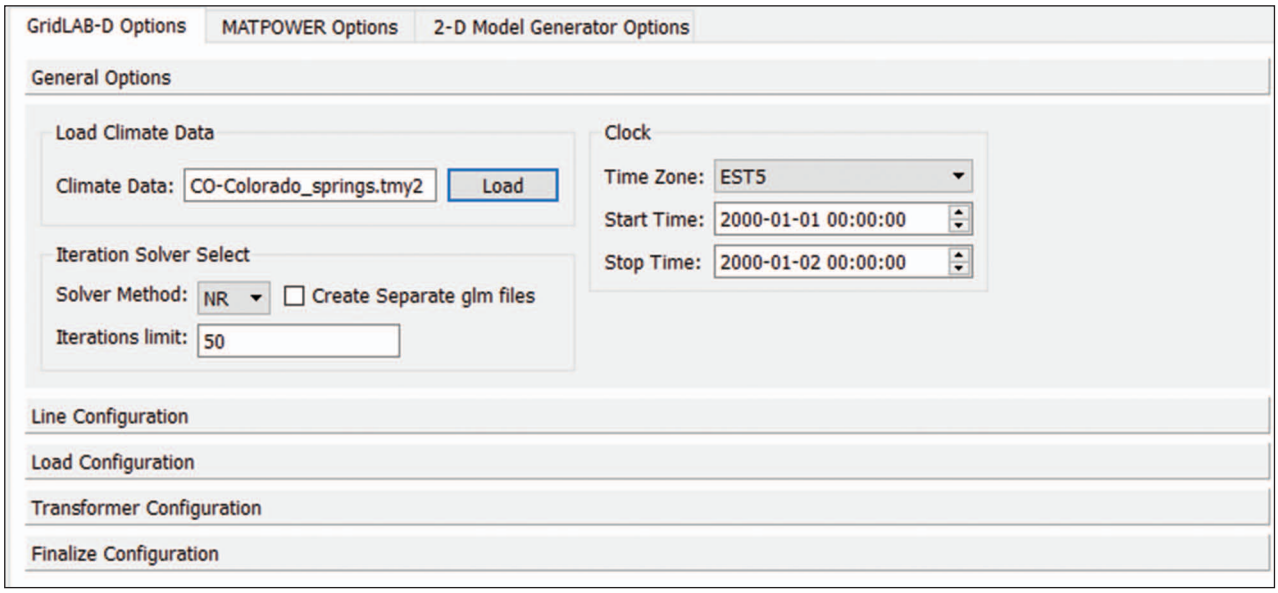

Figure 5. A sample of the graphical user interface for DEx.py.

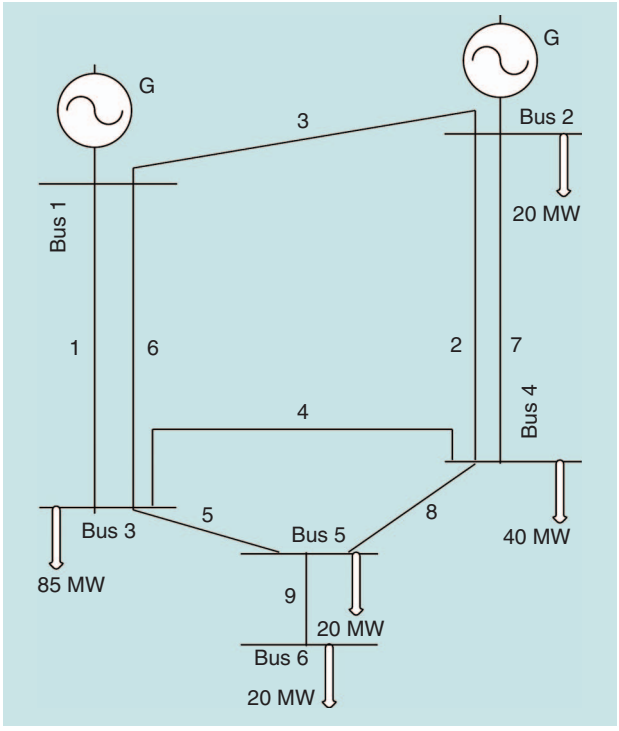

Figure 6. The Roy Billinton test system.

have at least one subfeeder (level 2). Each primary feeder and subfeeder consists of at least one load point, at level 3. Each load point may consist of additional subload points (level 4). The upper bounds for the number of primary feeders, subfeeders per primary feeder, load points per feeder, and additional subload points per load point per feeder for each transmission node are set by the user to create a new distribution topology.

The link between the primary feeders and subfeeders can be configured as an overhead line or an underground cable. The load points and subload points in the distribution topology represent the connection points for the low-voltage (LV) network where the residential loads are connected, as shown in Figure 8. The LV network consists of step-down distribution transformers, triplex nodes, and triplex lines for each of the three phases. The residential loads are connected at the end points of each phase of the LV network at distribution voltage level. At the time of writing, only residential homes are modeled in DEx.py; however, we are working to add commercial- and industrial-type loads as well. Once the distribution system network is generated, the load profiles need to be generated for each of the newly attached homes. This leads to the second step of DEx.py: end-user load population.

To split the total load at the PCC among individual house loads, we designed a combined top-down, bottom-up approach. First, the total bus load is split into the aggregate load of each individual house connected on a distribution network, using a stochastic approach (top down). One-dimensional random walk theory is used to generate the load profile for the residential loads that are distributed across a distribution feeder network. Here, the load profile for each residential load is scaled from a time-varying system load at the transmission node and used as the input for the one-dimensional random walk process. Load curves for a small subset of the residential loads that populate a GridLAB-D feeder are presented in Figure 9. Each house has a different, yet realistic, load curve; the ensemble sum of the residential load curves represents the load curve at the PCC.

To obtain the usage pattern of individual smart grid technologies at each household (bottom up), we use a method based on queueing theory. We model each household as an infinite capacity computing server with asset usage analogous to task arrivals. In the $M_t/G/\infty$ queue, applications arrive nonhomogeneously with Markovian distribution (i.e., time-varying Poisson process), generally distributed execution times, and infinite capacity. By

varying the rate parameter of a Poisson process, we can determine the usage pattern of individual smart grid technologies that sum to the aggregate house load. An example household load curve generated from this method is given in Figure 10. The combined top-down, bottom-up approach will ensure that hundreds of thousands of individual smart grid technologies will each operate in a way that match the desired system load behavior.

Control of Distributed Energy Resources in GridLAB-D Using Bus.py

Once the network and end-user models have been generated, the next step in the cosimulation is to enable communication between the tools, in this case MATPOWER and GridLAB-D. Bus.py is an abstract software transmission bus interface that facilitates the cosimulation of tools, e.g., customer HEMSs and the distribution feeder. The fine-grained modeling of the distribution system in GridLAB-D makes it an ideal tool to perform smart grid technology studies at the distribution level. Bus.py leverages the fine-grained modeling and interprocess communication of GridLAB-D to enable a myriad of future-grid scenarios.

The name Bus.py derives from the fact that its purpose is to emulate a transmission-level bus. To accomplish this goal, an abstract interface is provided in Python that includes interprocess communication to the distribution simulator GridLAB-D. To facilitate interactive simulation, GridLAB-D currently provides a Hypertext Transfer Protocol (HTTP) server for interprocess communication. Other implementations provided by the Bus.py interface are a constant load bus, time-series load bus, and resistance-based load bus (where a Thévenin equivalent resistance is used in conjunction with Ohm's law to determine the load at a bus). This section focuses on our implementation of our Bus.py interface with GridLAB-D.

The guiding principle of Bus.py is a flexible and easy-to-use interface to enable the cosimulation of electric power system tools. Pseudocode that describes the operation of Bus.py with a generic cosimulator (e.g., microgrid controller, HEMS controller) is presented in Figure 11. Bus.py has four main functions: `load_bus`, `start_bus`, `transaction`, and `stop_bus`.

The `load_bus` function reads from a bus input file all relevant parameters for Bus.py to be used during the cosimulation process. The input file will specify which type of bus will be modeled (e.g., a GridLAB-D bus), simulation time information (i.e., start time, stop time, and time step), and any other relevant parameters for that bus type. The input file is specified using the JavaScript Object Notation, an easy-to-read set of key-value pair strings. `Load_bus` will return a `Bus` object to be used for the cosimulation. Once a `Bus` object is loaded and the cosimulator is initialized, `start_bus` will commence the bus cosimulation environment. In the case of a GridLAB-D bus, this will initiate a GridLAB-D simulator process and start its HTTP server for interprocess communication with the Bus.py interface.

After the cosimulation environment is commenced with `start_bus`, the main simulation loop begins.

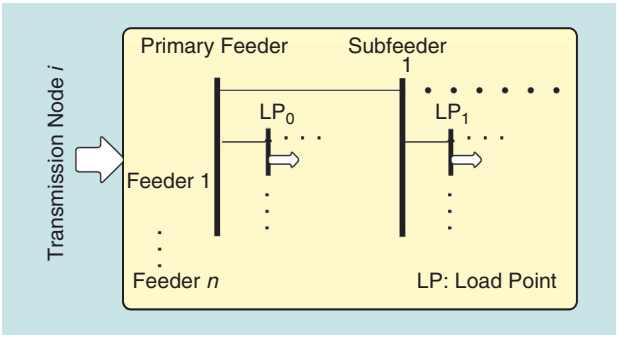

Figure 7. Connection of GridLAB-D feeders to the Roy Billinton test system transmission bus.

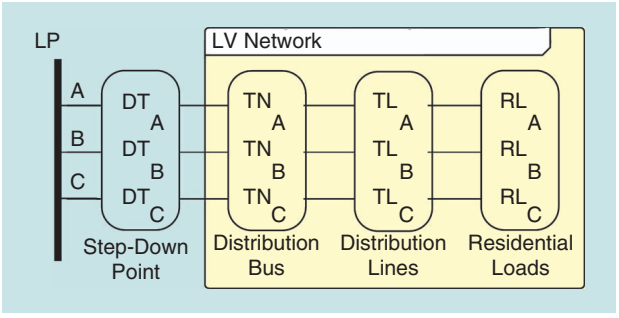

Figure 8. The load points connected to the GridLAB-D feeder model.

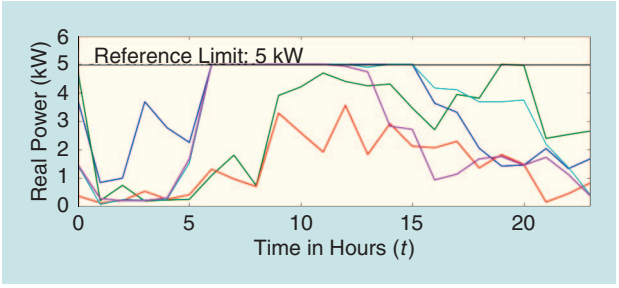

Figure 9. A subset of load curves for individual houses on the GridLAB-D feeder.

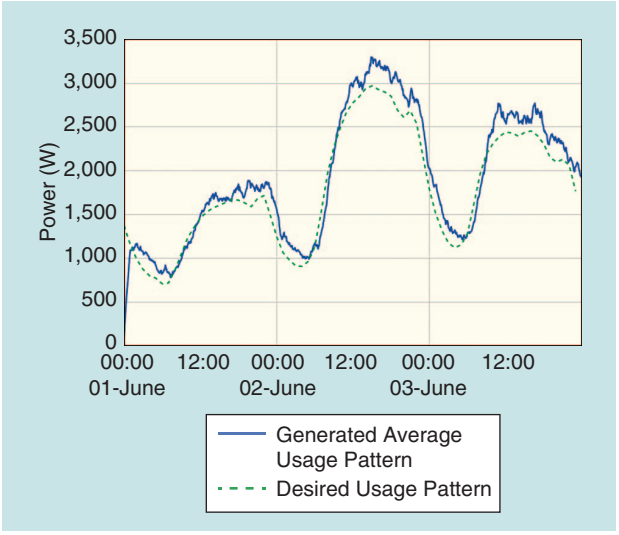

Figure 10. An example usage pattern generated by the $M_t/G/\infty$ queue model for a single home. The dashed green line represents the desired load from the top-down approach. The solid blue line is the generated usage pattern averaged over 500 samples.

```
1: Bus = load_bus(input_file)
2: cosimulator.initialize()
3: Bus.start_bus()
4: repeat
5:     bus_inputs = cosimulator.optimize()
6:     bus_outputs = Bus.transaction(bus_inputs)
7:     cosimulator.process(bus_outputs)
8: until Bus.finished
9: Bus.stop_bus()
10: cosimulator.postprocess()
```

Figure 11. Bus.py pseudocode with an abstract cosimulator process.

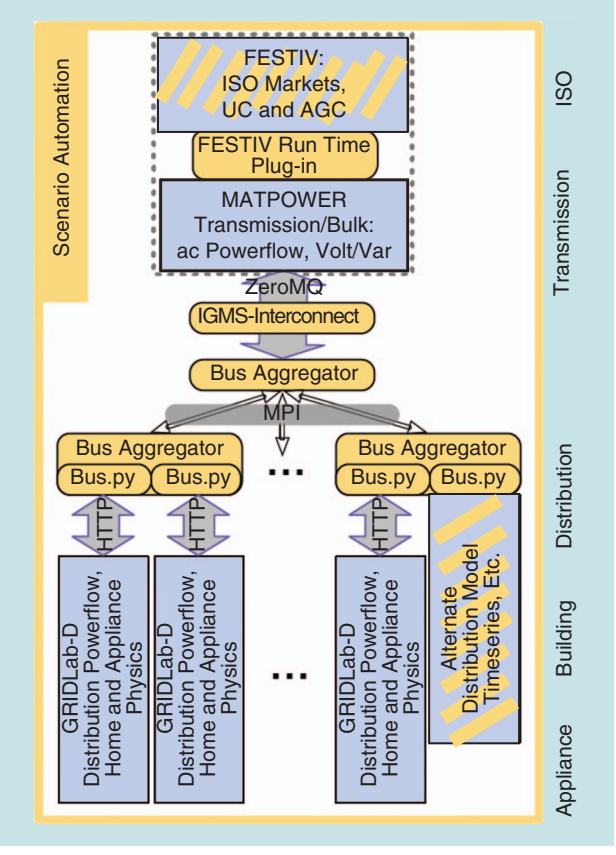

Figure 12. The IGMS HPC-enabled cosimulation environment.

Each iteration of the loop represents one time step in the simulation. The basic order of operations at time step *t* is: 1) obtain the inputs to the `Bus` for time step *t* from the cosimulator, 2) perform a transaction with `Bus`, and 3) process the outputs of `Bus` using the cosimulator. The `transaction` function passes the inputs to the `Bus`, steps time forward, and returns the specified outputs. For GridLAB-D, this will 1) send key-value pairs (e.g., customer1.load = 10 kW) to GridLAB-D using HTTP, 2) step GridLAB-D forward one simulation time step, and 3) request and return the GridLAB-D simulated output (e.g., substation power). The single `transaction` function simplifies the communication with GridLAB-D to present a powerful cosimulation tool.

After the stop time of the simulation is reached, the simulation loop will end. The `stop_bus` function will stop any associated processes/files used by the `Bus` object (e.g., stop the GridLAB-D process in the case of a GridLAB-D bus). Once the main simulation loop ends and `Bus` is stopped, it may be useful to perform postprocessing with the cosimulator, such as visualization of the time-series output.

Putting It All Together: IGMS

IGMS is an HPC-enabled cosimulation environment that focuses on bulk power market and technical operation impacts of distributed energy resources developed at NREL. Specifically, IGMS simulates thousands of individual distribution systems under the purview of an ISO with detailed market operation and automatic generator control (AGC)-level reserve deployment. The thousands of distribution systems are simulated using GridLAB-D and interfaced using the Bus.py tool to the bulk power level, simulated using a combination of MATPOWER and Flexible Energy Scheduling Tool for Integration of Variable Generation (FESTIV), a model of the bulk power market with an accurate day-ahead and real-time unit commitment and economic dispatch model, including AGC control. The IGMS hierarchical framework, presented in Figure 12, mirrors the structure of the power grid with a set of integrated bulk-level models interacting with thousands of distribution feeder models. Initial studies using IGMS hint at the need for cosimulation to determine the potential impacts and interaction of increasingly widespread distributed energy resources with the bulk power system.

Case Studies

To showcase the usefulness of cosimulation, two case studies are presented, both using an aggregator-based residential demand-response program (given in Figure 13) and Bus.py interacting with GridLAB-D. On the right-hand side of Figure 13 is the traditional power system and market structure depicting the ISO to the distribution system operator (DSO) for delivering electricity to the residential customer. The left-hand side of Figure 13 summarizes the residential demand-response program. The demand-response exchange (DRX) is an ancillary market in a fully deregulated market structure that provides demand-response services to the ISO. The aggregator is a for-profit market entity engaged in interacting with the customer and the bulk power market in a fully deregulated market structure. The aggregator coordinates a set of participating customers, each with a set of smart grid distributed energy resource assets, and brings the result (e.g., load reduction) to the DRX for market exchange.

The first illustrative example involves peak-load minimization through load shifting using Bus.py. The resulting substation apparent power throughout the simulation period is presented in Figure 14. Because the objective function in the optimization problem was peak reduction, the peak load between 3:00 and 6:00 p.m. is shown in more detail in the inset of Figure 14. The solid blue line represents the baseline load of the system (i.e., the system load in the absence of the aggregator demand

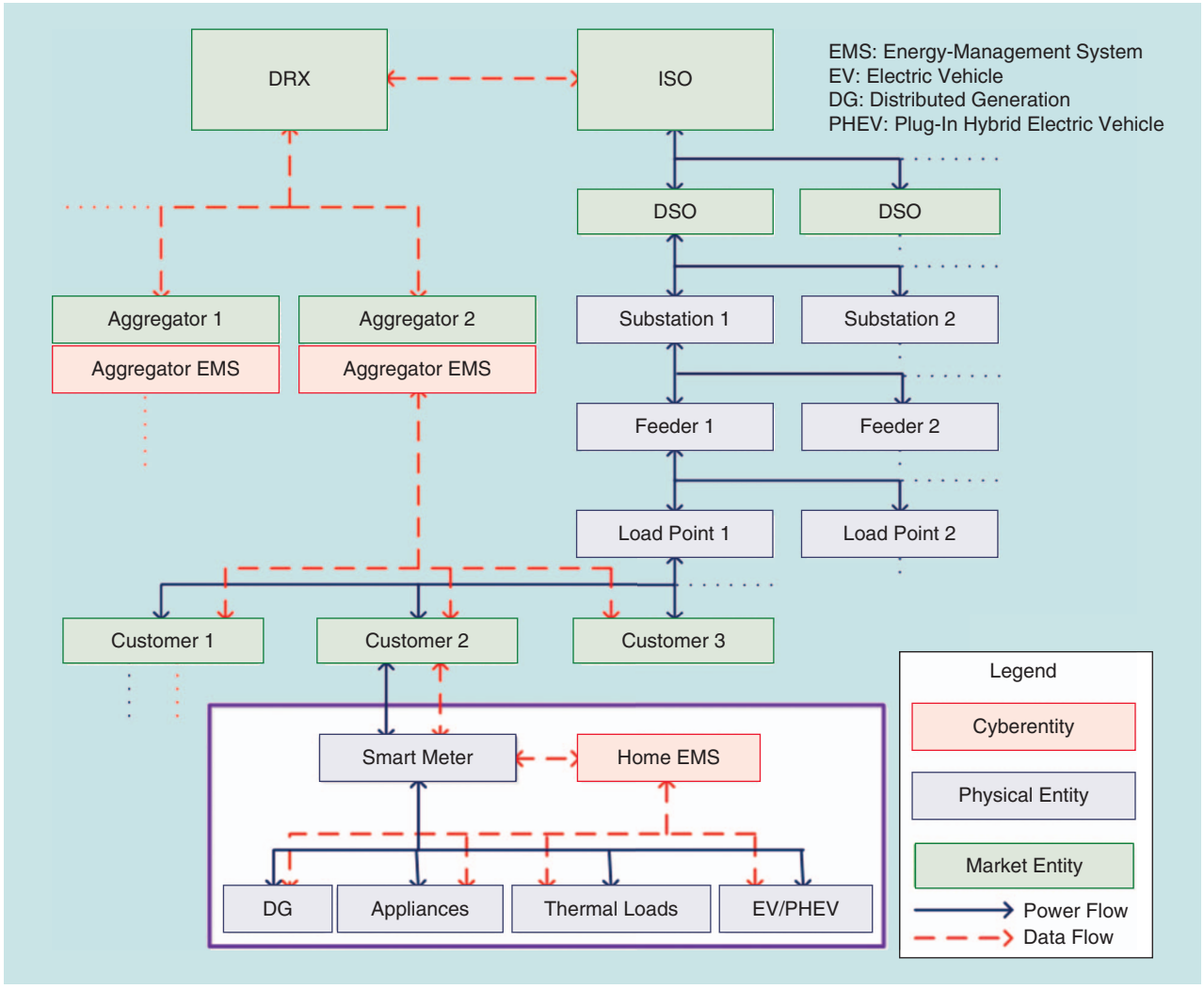

Figure 13. The architecture and communication for the aggregator-based residential demand-response program.

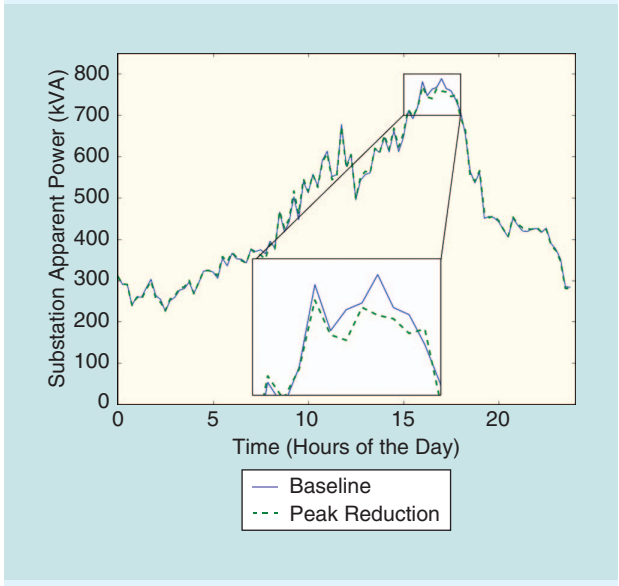

Figure 14. A comparison of the substation apparent power, in kilovolt-amperes, between the baseline (solid blue line) and aggregator demand-response (dashed green line) cases resulting from peak minimization.

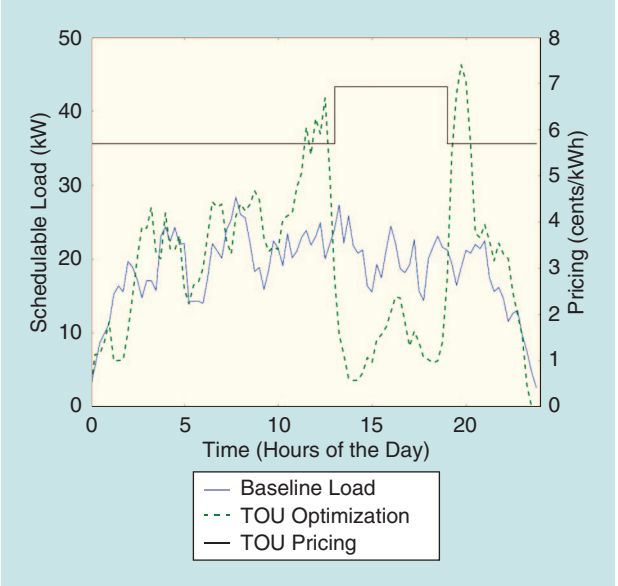

Figure 15. A comparison between the customer schedulable load of the baseline (solid blue line) and aggregator demand-response (dashed green line) cases resulting from customer cost minimization.

response). The dashed green line represents the substation apparent power, in kilovolt-amperes, after the aggregator-based residential demand response was performed. The peak system power at 5:15 p.m. was reduced by 19.2 kVA, the total schedulable load available for demand response at that time. This corresponds to a 2.5% decrease in peak system load, which aligns with the U.S. Federal Energy Regulatory Commission expectations for demand response in the residential sector.

The second example involves total customer cost minimization of the schedulable demand-response loads (i.e., smart appliances) in a time-of-use (TOU) market, data used from Duke Energy in North Carolina, using Bus.py. The resulting load of the controllable assets, which represents 450 kWh of the distribution feeder load, throughout the simulation period is presented in Figure 15. The solid blue line represents the baseline schedulable loads of the customers. The dashed green line represents the customer schedulable loads, in kilowatts, after the aggregator-based residential demand response was performed while optimizing for the minimization of customer energy cost. The solid black curve shows the TOU pricing used in the optimization, with the corresponding *y*-axis values on the right, in cents/kWh.

During the TOU peak-pricing period, from 1:00 to 7:00 p.m. (hours 13 to 19), the total schedulable loads of all the customers were pushed off peak and resulted in a reduction from 123 to 52 kWh. This makes sense because the only way to reduce customer cost in the TOU pricing scheme is to move load from on peak to off peak. The reason that all schedulable load was not moved off peak is because of the customer-defined comfort constraints. It is interesting to note the resulting rebound effect—the change in the consumption pattern of electricity from the changing cost of electricity—that occurs on either side of the transition from off-peak to on-peak pricing at 1:00 p.m. and at 7:00 p.m.

Conclusions

As more smart grid technologies are implemented in the distribution system, the infeasibility of a single tool simulating different electric power system domains at a detailed level limits the ability to properly study and quantify their impacts. DEx.py and Bus.py enable the dynamic integration of multiple electric power system simulation tools, such as GridLAB-D and MATPOWER. The use of DEx.py, Bus.py, and IGMS enables the cosimulation of transmission and distribution systems, customer HEMSs and the distribution system, as well as many other use cases that cross traditional electric power system domain boundaries, leading to the rapid design and development of the technologies and controls required for the future electric power grid. Work is ongoing to link DEx.py, Bus.py, and the constituent simulation environments to demonstrate simulations of large systems representative of smart cities connect to the bulk electricity grid.

Disclaimer

Material presented here is an anthology of research works of the authors published or under review consideration by the IEEE, identified in the "For Further Reading" section (denoted by "[a]").

For Further Reading

[a]T. M. Hansen, B. Palmintier, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel, "Bus.py: A GridLAB-D communication interface for smart distribution grid simulations," in *Proc. IEEE Power & Energy Society General Meeting*, July 2015, pp. 1–5.

[a]T. M. Hansen, R. Roche, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel, "Heuristic optimization for an aggregator-based resource allocation in the smart grid," *IEEE Trans. Smart Grid*, vol. 6, no. 4, pp. 1785–1794, July 2015.

D. P. Chassin, K. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *Proc. IEEE Power & Energy Society Transmission and Distribution Conf. and Expo.*, Apr. 2008, pp. 1–5.

R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and educations," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

R. Billinton and S. Jonnavithula, "A test system for teaching overall power system reliability assessment," *IEEE Trans. Power Syst.*, vol. 11, no. 4, pp. 1670–1676, Nov. 1996.

[a]T. M. Hansen, E. K. P. Chong, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel, "A partially observable Markov decision process approach to residential home energy management," submitted for publication.

[a]R. Kadavil, T. M. Hansen, and S. Suryanarayanan, "An algorithmic approach for creating diverse stochastic feeder datasets for power systems co-simulations," submitted for publication.

[a]B. Palmintier, E. Hale, T. M. Hansen, W. Jones, D. Biagioni, H. Sorensen, and B.-M. Hodge, "IGMS: An integrated ISO-to-appliance scale grid modeling system," submitted for publication.

Biographies

Timothy M. Hansen (timothy.hansen@sdstate.edu) is an assistant professor in the Electrical Engineering and Computer Science Department at South Dakota State University.

Rahul Kadavil (rahul.kadavil@colostate.edu) is a graduate student with the Department of Electrical and Computer Engineering at Colorado State University.

Bryan Palmintier (bryan.palmintier@nrel.gov) is a senior research engineer in the Power Systems Engineering Center at the U.S. National Renewable Energy Laboratory.

Siddharth Suryanarayanan (sid@colostate.edu) is an associate professor in the Department of Electrical and Computer Engineering at Colorado State University.

Anthony A. Maciejewski (aam@colostate.edu) is a professor and the head of the Department of Electrical and Computer Engineering at Colorado State University.

Howard Jay Siegel (hj@colostate.edu) is the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University, where he is also a professor of computer science.

Edwin K.P. Chong (edwin.chong@colostate.edu) is a professor in the Department of Electrical and Computer Engineering and Department of Mathematics at Colorado State University.

Elaine Hale (elaine.hale@nrel.gov) is a senior engineer in the Strategic Energy Analysis Center at the U.S. National Renewable Energy Laboratory.