Bus.py: A GridLAB-D Communication Interface for Smart Distribution Grid Simulations

Timothy M. Hansen, *Graduate Student Member, IEEE*, Bryan Palmintier, *Member, IEEE*, Siddharth Suryanarayanan, *Senior Member, IEEE*, Anthony A. Maciejewski, *Fellow, IEEE*, and Howard Jay Siegel, *Fellow, IEEE*

***Abstract*—As more Smart Grid technologies (e.g., distributed photovoltaic, spatially distributed electric vehicle charging) are integrated into distribution grids, static distribution simulations are no longer sufficient for performing modeling and analysis. GridLAB-D is an agent-based distribution system simulation environment that allows fine-grained end-user models, including geospatial and network topology detail. A problem exists in that, without outside intervention, once the GridLAB-D simulation begins execution, it will run to completion without allowing the real-time interaction of Smart Grid controls, such as home energy management systems and aggregator control. We address this lack of runtime interaction by designing a flexible communication interface, Bus.py (pronounced bus-dot-pie), that uses Python to pass messages between one or more GridLAB-D instances and a Smart Grid simulator. This work describes the design and implementation of Bus.py, discusses its usefulness in terms of some Smart Grid scenarios, and provides an example of an aggregator-based residential demand response system interacting with GridLAB-D through Bus.py. The small scale example demonstrates the validity of the interface and shows that an aggregator using said interface is able to control residential loads in GridLAB-D during runtime to cause a reduction in the peak load on the distribution system in (a) peak reduction and (b) time-of-use pricing cases.**

***Index Terms*—aggregator, communication interface, demand response, energy systems integration, GridLAB-D, simulation test bed.**

I. Introduction

In Title XIII of the Energy Independence and Security Act of 2007, the U.S. Congress set forth the tenets of modernizing the electricity grid through the Smart Grid initiative [1]. Some of these tenets include the deployment, development, and integration of distributed energy resources, "smart" technologies and appliances, and advanced storage devices. The integration of these technologies requires new modeling and simulation tools, but it is difficult for a single tool to model all power systems domains in adequate detail. This leads to the use of *co-simulation* tools where multiple individual tools, each modeling a single domain in detail, interact while running simultaneously [2].

This work introduces Bus.py[1] (pronounced bus-dot-pie), an abstract software transmission bus interface that facilitates the co-simulation of tools, e.g., customer home energy management systems (HEMS) and the distribution feeder. One such simulation tool is GridLAB-D: a flexible distribution system simulator that allows fine-grain modeling of distribution assets from the substation transformer down to the individual household [3]. This fine-grained modeling of the distribution system makes it an ideal tool to perform Smart Grid technology studies at the distribution level. Bus.py leverages the fine-grained modeling and inter-process communication of GridLAB-D to enable a myriad of future-grid scenarios. To demonstrate Bus.py's effectiveness for performing co-simulation studies, we present two illustrative examples.

In the last few years, there has been relevant related work in the area of electric power grid co-simulation tools, specifically GridLAB-D. GridMat is a Matlab-GridLAB-D interface for residential controllers for use in a residential microgrid [4]. GridMat includes a similar GridLAB-D control interface for residential control only. In addition to residential control, Bus.py enables additional use cases, such as transmission-distribution integration through the use of high-performance computing. In [5], another Matlab-GridLAB-D co-simulation framework was introduced. Our work differs in that Bus.py offers dynamic interaction with GridLAB-D while [5] performs all optimization offline to be used as a static GridLAB-D simulation. GridSpice [6] is a GridLAB-D cloud infrastructure that enables the deployment of many GridLAB-D instances to the Amazon cloud for the purpose of transmission-level power analysis with MATPOWER [7]. In contrast, Bus.py allows the dynamic simulation of multiple grid use-cases without relying on a cloud infrastructure, but can be enabled by many computing platforms (e.g., personal computer, high-performance computing). In this work, we make the following unique contributions:

(a) the design of Bus.py, a software transmission bus interface for use in Smart Grid co-simulation studies,
(b) the creation of a flexible communication interface with the distribution simulator GridLAB-D,
(c) a discussion of the usefulness of the interface, and
(d) a demonstration of Bus.py interacting with GridLAB-D simulating a small set of customers on a distribution feeder and an aggregator entity.

The rest of this paper is organized as follows. Section II describes the Bus.py interface. The usefulness of Bus.py is presented in Section III by illustrating how one can simulate the control of multiple residential households with a residential aggregator using Bus.py and GridLAB-D. Concluding remarks are given in Section IV.

II. Bus.py

A. Overview

The name Bus.py derives from the fact that its purpose is to emulate a transmission-level bus. To accomplish this goal,

T. M. Hansen and B. Palmintier are with the National Renewable Energy Laboratory, Golden, CO, 80401 USA, e-mail: tim.hansen@nrel.gov, bryanp@ieee.org.

T. M. Hansen, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel are with Colorado State University, Fort Collins, CO, 80523 USA, e-mail: {timothy.hansen,sid,aam,hj}@colostate.edu.

This work was supported by the U.S. Department of Energy under Contract No. DE-AC36-08-GO28308 with the National Renewable Energy Laboratory.

[1]See Bus.py project updates at http://www.engr.colostate.edu/sgra/. Any questions or comments may be directed to Timothy M. Hansen.

978-1-4673-8040-9/15/$31.00 ©2015 IEEE

an abstract interface is provided in Python that includes inter-process communication to the distribution simulator GridLAB-D. To facilitate interactive simulation, GridLAB-D currently provides an HTTP server for inter-process communication. Other provided implementations of our 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 paper focuses on our implementation of our Bus.py interface with GridLAB-D, described in the following subsection. A few interesting co-simulation use cases, made possible with Bus.py, are given in Subsection II-C.

B. Interface

The principle of Bus.py is a flexible, easy-to-use interface to enable the co-simulation of electric power system tools. Pseudocode that describes the operation of Bus.py with a generic co-simulator (e.g., microgrid controller, HEMS controller) is presented in Fig. 1. Bus.py has four main functions: load_bus, start_bus, transaction, and stop_bus (given as lines 1, 3, 6, and 9 in Fig. 1, respectively), each described in detail below.

```
 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()
```

Fig. 1. Bus.py pseudocode with an abstract co-simulator process.

The **load_bus** function reads from a bus input file all relevant parameters for Bus.py to be used during the co-simulation 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 timestep), and any other relevant parameters for that bus type. The input file is specified using the JavaScript Object Notation (JSON), an easy-to-read set of key-value pair strings. Load_bus will return a Bus object to be used for the co-simulation.

Once a Bus object is loaded and the co-simulator is initialized, **start_bus** will start the bus co-simulation environment. In the case of a GridLAB-D bus, this will start a GridLAB-D simulator process and start its HTTP server for inter-process communication with the Bus.py interface.

After the co-simulation environment is started with start_bus, the main simulation loop begins (lines 4–8 in Fig. 1). Each iteration of the loop represents one timestep in the simulation. The basic order of operation at timestep t is: (1) obtain the inputs to the Bus for timestep t from the co-simulator, (2) perform a transaction with Bus, and (3) process the outputs of Bus using the co-simulator. 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 timestep, 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 co-simulation tool.*

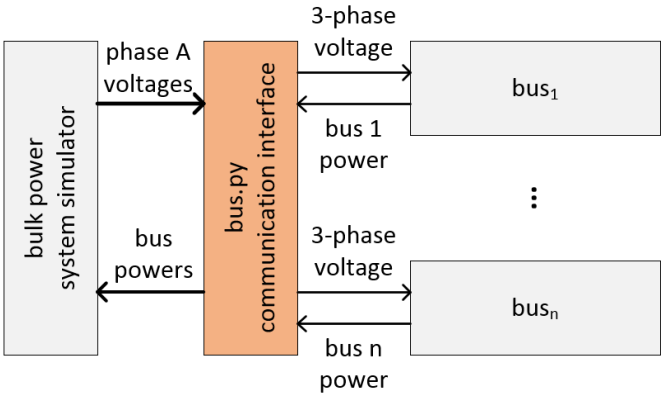

Fig. 2. Bus.py interfacing a bulk power simulator, such as MATPOWER [7], and many Bus instances (e.g., GridLAB-D, time-series). This scenario could be used to integrate transmission and distribution system simulators.

After the stop time of the simulation is reached (line 8 in Fig. 1), 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 post-processing with the co-simulator, such as visualization of the time-series output.

C. Use Cases

The following subsection will describe a non-exhaustive list of possible use cases for Bus.py-enabled co-simulation. The first use case is the co-simulation of a transmission-level simulator, such as MATPOWER [7], and a Bus instance at each bus in the transmission system. This interaction is depicted in Fig. 2. The physical simulation of the bulk power system and distribution systems have traditionally been modeled as separate with simplified representations of their interaction in each individual simulation. As more Smart Grid technologies (e.g., demand response technologies, distributed energy resources) are implemented in the distribution system, these separate simulations may no longer be adequate, such as in the case with bi-directional power flow resulting from a high penetration of distributed photovoltaic generation [8]. *Bus.py enables bi-directional power flow studies resulting from increased distributed energy resources.*

A second use case is presented in Fig. 3. In this use case, many customer HEMS are optimizing and interacting on a single distribution feeder, modeled by GridLAB-D. Because each house is located at a separate physical location on the distribution feeder, each will have slightly different physical interactions with said distribution feeder (e.g., different voltage levels, different phases). Increasing the number of integrated Smart Grid technologies and allowing retail customers direct access to market prices may increase the price-elasticity of demand, leading to an increased volatility in power systems [9]. If many HEMS are simulated individually (i.e., without modeling the effect of the distribution grid), this volatility may not be properly modeled. *Bus.py allows the study of the physical responses on distribution feeders from a multitude of customer*

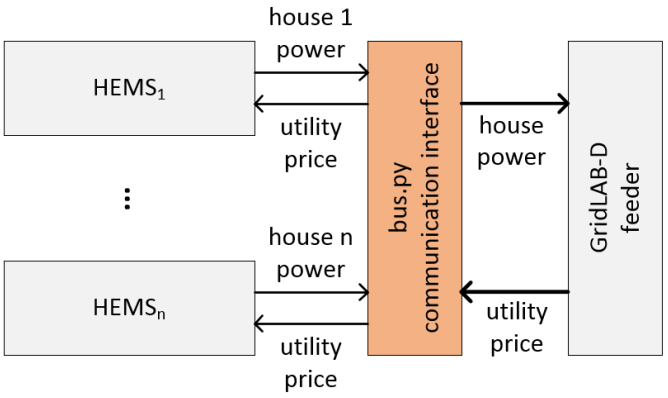

Fig. 3. Bus.py interfacing a GridLAB-D feeder with many customer home energy management systems (HEMS). This scenario could be used to determine the effect of many HEMS on a distribution system.

HEMS and other Smart Grid technologies. Additionally, because the distribution system properties (e.g., voltage, retail prices in a real-time pricing environment) change through time, a static simulation of customer HEMS is not sufficient. *Bus.py facilitates the dynamic co-simulation of Smart Grid technologies and their respective distribution feeder.*

This list of use cases is not exhaustive but is meant to illustrate the usefulness of the Bus.py interface to enable the co-simulation of tools. Additional use cases could include market (with a bulk power market simulator such as FESTIV [10]) and communication (with a communication simulator such as ns-3 [11]) co-simulations.

III. Residential Aggregator Demand Response

A. System Model

To demonstrate the usefulness of the Bus.py interface, we present two illustrative co-simulation examples. The first is a common load shifting problem to reduce the peak system load, presented in Subsection III-C. The second example quantifies the effect of time-of-use (ToU) pricing on the system load profile, presented in Subsection III-D. In both examples, an aggregator-based residential demand response is used, described in detail below and presented in Fig. 4.

On a given distribution feeder, we model $\boldsymbol{N}$ customer households. Each household i has a set of $\boldsymbol{n_i}$ loads (e.g., smart appliances) that are made available to an aggregator for rescheduling. For each load j of customer i, the aggregator is assumed to know:

- $\boldsymbol{p_{ij}}$ – the average load, in kW,
- $\boldsymbol{\delta_{ij}}$ – the duration of the load's operation,
- $\boldsymbol{A_{ij,min}}$ – the start of the customer-defined rescheduling window, and
- $\boldsymbol{A_{ij,max}}$ – the end of the customer-defined rescheduling window.

The time period from $A_{ij,min}$ to $A_{ij,max}$ is a customer-defined rescheduling window that is used to take into account customer comfort [12]. The aggregator-based residential demand response is set up as an optimization problem. For each customer load, the aggregator must find the rescheduled time, $\boldsymbol{T_{ij}}$, subject to $A_{ij,min} \leq T_{ij} < A_{ij,max}$, that optimizes an objective function (described in Subsections III-C and III-D).

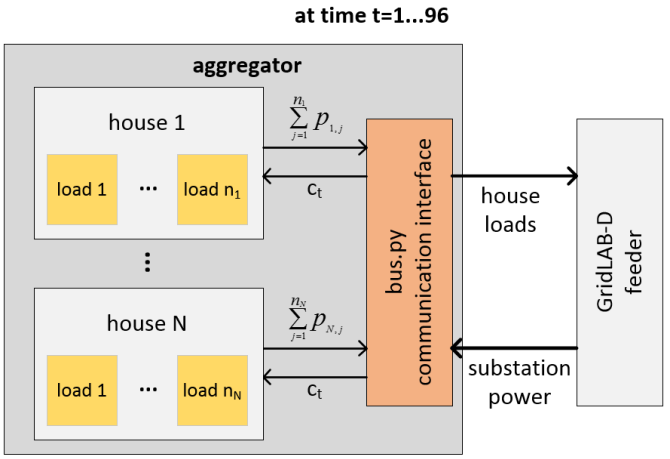

Fig. 4. The proposed aggregator system model. At each 15-minute time step, the aggregator determines the total schedulable load at each customer household and passes the information of the loads to GridLAB-D through the Bus.py interface. The aggregator then requests the substation apparent power following the scheduling of the loads to verify and quantify the change in load.

To perform the optimization, a genetic algorithm is used, described in the following subsection.

The simulation is set up for a 24-hour period with a time step, $\boldsymbol{\Delta t}$, of 15-minutes (i.e., 96 simulation periods) occurring on June 1, 2012. For the GridLAB-D inputs, 206 houses were placed on a Pacific Northwest National Laboratory taxonomy feeder (using the method from [13]) representing a lightly populated rural area (R4-25.00-1) [14]. Weather data for Charlotte, North Carolina, was used to match the location of the taxonomy feeder in the non-coastal southeast United States. Each customer has between one and four schedulable loads leading to a total of 543 loads. If we let $\mathcal{N}(\mu, \sigma)$ represent a normal distribution with mean μ and standard deviation σ, then each load has the duration and power generated randomly from $\mathcal{N}(4, 2)$ and $\mathcal{N}(0.8, 0.2)$, respectively. A baseline load is determined by randomly assigning each load to start at one of the 96 simulation periods. The duration of the customer-defined rescheduling window is randomly generated around this baseline time for each load from $\mathcal{N}(16, 4)$. Note that these values are used for simulation purposes only and do not have any bearing on the usefulness of the Bus.py interface.

B. Heuristic Approach

In this example, a Genitor [15] version of genetic algorithm (GA) implemented in Python was used to perform the aggregator optimization in conjunction with Bus.py. We used a GA as an example global search heuristic that has been shown to work well in many optimization problems in power systems, such as economic dispatch [16] and unit commitment [17]. A gene within the chromosome represents an individual schedulable load. Let $\boldsymbol{s_{ij}}$ be a real value in the interval $[0, 1]$ representing the T_{ij}. To obtain the time interval that each load was scheduled, the following equation was used: $T_{ij} = A_{ij,min} + s_{ij}\,(A_{ij,max} - A_{ij,min})$. The $[0, 1]$ representation of s_{ij} was used to avoid violating the customer-defined rescheduling window constraint of the loads [18].

The Genitor is a steady-state algorithm that maintains a ranked list of chromosomes, leading to implicit elitism, i.e.,

between generations, the best solutions are kept. In each generation, two parents are selected using the linear bias function (as defined in [15]) for crossover. The linear bias selection function requires a linear bias parameter that is a real value in the interval $(1, 2]$. A linear bias parameter of 1.5 means the best-ranked solution has a 50% greater chance of being selected than the median solution. A two-point crossover method is used. The mutation operator on a gene will randomly generate a new s_{ij} between $[0, 1]$. In this simulation study, the genetic algorithm runs for 10,000 iterations using 50 chromosomes with a probability of mutation of 0.05 and a linear bias of 1.5. Note that these values are used for simulation purposes only and do not have any bearing on the usefulness of the Bus.py interface.

C. Peak Load Minimization

The first illustrative example involves peak load minimization through load shifting using Bus.py as in Fig. 4. Load or demand shifting is a well-known demand response technique [19], [20]. Let $\boldsymbol{\kappa_t}$ be the known fixed load of the distribution system at time t in kW (i.e., the unschedulable load). We define the peak load version of the aggregator-based residential demand response problem as: find $T_{ij} \forall i, j$, subject to $A_{ij,min} \leq T_{ij} < A_{ij,max}$, to minimize:

$$\max_{t=1\ldots96} \kappa_t + \sum_{i=1}^{N} \sum_{j=1}^{n_i} \begin{cases} p_{ij} & T_{ij} \leq t < (T_{ij} + \delta_{ij}) \\ 0 & else \end{cases} . \quad (1)$$

The resulting substation apparent power throughout the simulation period is presented in Fig. 5. Because the optimization was peak reduction, the peak load between 3:00 to 6:00 pm is shown in more detail in the inset in Fig. 5. The solid blue line represents the baseline load of the system (i.e., the system load in the absence of the aggregator demand response). The dashed green line represents the substation apparent power, in kVA, after the aggregator-based residential demand response was performed. The peak system power at 5:15 pm 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 Federal Energy Regulatory Commission (FERC) expectations for demand response in the residential sector [21].

D. Customer Cost Minimization

The second illustrative example involves total customer cost minimization of the schedulable demand response loads (i.e., smart appliances in this work) in a ToU market using Bus.py as in Fig. 4. The effect of ToU on load profiles is a common optimization problem [22]. Let $\boldsymbol{c_t}$ be the cost of electricity at time t in \$/kWΔt (where Δt was 15-minutes). Because the distribution feeder modeled is using inputs for North Carolina, the ToU pricing from Duke Energy in North Carolina was used [23]. Note that only the energy charge was considered. Modeling the monthly demand charge would require a longer simulation period, which is possible using Bus.py but not shown here for brevity. We define the energy cost minimization version of the aggregator-based residential demand response problem as: find $T_{ij} \forall i, j$, subject to $A_{ij,min} \leq T_{ij} < A_{ij,max}$, to minimize:

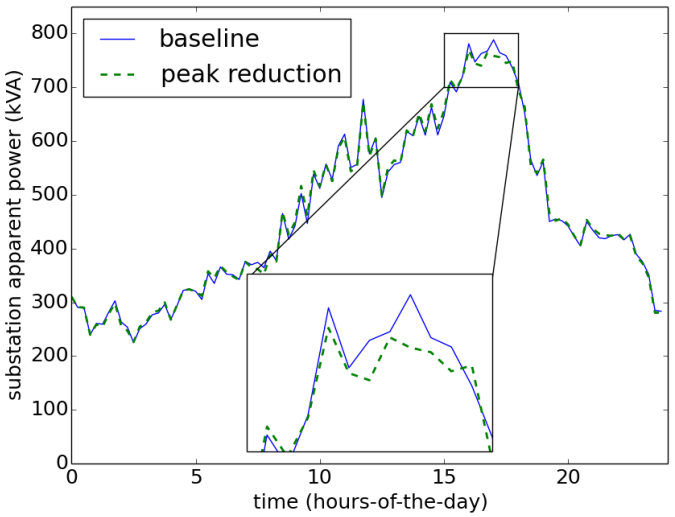

Fig. 5. The comparison of the substation apparent power, in kVA, between the baseline (solid blue line) and aggregator demand response (dashed green line) cases resulting from peak minimization. Because the optimization was peak minimization, the peak of the system is shown in more detail in the inset.

$$\sum_{t=1}^{96} c_t \sum_{i=1}^{N} \sum_{j=1}^{n_i} \begin{cases} p_{ij} & T_{ij} \leq t < (T_{ij} + \delta_{ij}) \\ 0 & else \end{cases} . \quad (2)$$

The resulting load of *only the schedulable loads*, which represents 450 kWh of the distribution feeder, throughout the simulation period is presented in Fig. 6. The solid blue line represents the baseline schedulable loads of the customers. The dashed green line represents the customer schedulable loads, in kW, 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 pm, the total schedulable loads of all the customers pushed off-peak and resulted in a reduction from 123 kWh 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 rescheduling window, from $A_{ij,min}$ to $A_{ij,max}$. It is interesting to note the resulting rebound effect—the change in the consumption pattern of electricity from the changing cost of electricity [24]—that occurs on either side of the transition from off-peak to on-peak pricing at 1:00 pm and at 7:00 pm. In our problem, the total energy remains the same because the loads are just shifted, but if thermal loads were considered sometimes *more* or *less* total energy is consumed because of this effect [25].

IV. Conclusions

Bus.py is an abstract software transmission bus interface that facilitates the co-simulation of electric power system tools. As more Smart Grid technologies are implemented in the distribution system, it becomes infeasible for a single tool to simulate all electric power system domains at a detailed level. Bus.py enables the dynamic simulation of multiple electric power systems tools, such as GridLAB-D and customer home energy management systems. We described the Bus.py

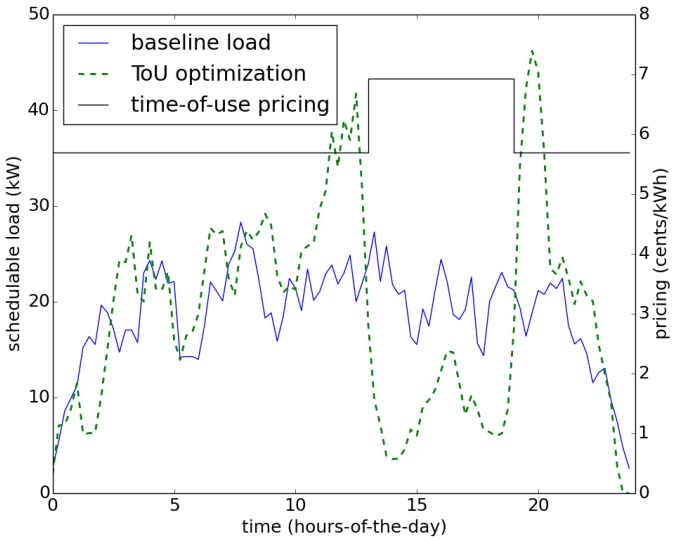

Fig. 6. The comparison between the load, in kW, of the customer schedulable loads made available to the aggregator between the baseline (solid blue line) and aggregator demand response (dashed green line) cases resulting from customer cost minimization. The time-of-use pricing used is given as the solid black curve.

interface interaction with GridLAB-D, a distribution system simulator. The existence of Bus.py enables the co-simulation of transmission and distribution systems, customer home energy management systems and the distribution system, as well as many other use cases. A demonstration of residential demand response resulting from multiple residential homes on a single distribution feeder through an aggregator in a single distribution area was achieved using the new Bus.py interface with GridLAB-D. The demonstration consisted of two examples: (a) system peak minimization and (b) customer cost savings in a time-of-use pricing scheme. The residential demand response resulted in (a) a 2.5% peak reduction and (b) an on-peak reduction from 123 kWh to 52 kWh, respectively.

Acknowledgments

The authors would like to thank Elaine Hale and the Distributed Energy Systems Integration group at the National Renewable Energy Laboratory for their valuable comments.

References

[1] 110th Congress of the United States of America, Energy Independence and Security Act of 2007, "Title XIII – Smart Grid," Dec. 2007.

[2] R. Roche, S. Natarajan, A. Bhattacharyya, and S. Suryanarayanan, "A framework for co-simulation of AI tools with power systems analysis software," in *23rd International Workshop on Database and Expert Systems Applications (DEXA)*, Sep. 2012, pp. 350–354.

[3] D. P. Chassin, K. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *IEEE PES Transmission and Distribution Conference and Exposition*, Apr. 2008, 5 pp.

[4] M. A. Al Faruque and F. Ahourai, "GridMat: Matlab toolbox for GridLAB-D to analyze grid impact and validate residential microgrid level energy management algorithms," in *IEEE PES Innovative Smart Grid Technologies Conference (ISGT 2014)*, 2014, 5 pp.

[5] D. Wang, B. de Wit, S. Parkinson, J. Fuller, D. Chassin, and C. Crawford, "A test bed for self-regulating distribution systems: Modeling integrated renewable energy and demand response in the GridLAB-D/MATLAB environment," in *IEEE PES Innovative Smart Grid Technologies Conference (ISGT 2012)*, 2012, 7 pp.

[6] K. Anderson, J. Du, A. Narayan, and A. E. Gamal, "GridSpice: A distributed simulation platform for the smart grid," in *IEEE 2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, May 2013, 5 pp.

[7] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[8] K. Turitsyn, P. Šulc, S. Backhaus, and M. Chertkov, "Distributed control of reactive power flow in a radial distribution circuit with high photovoltaic penetration," in *IEEE PES General Meeting 2010*, July 2010, 6 pp.

[9] M. Roozbehani, M. A. Dahleh, and S. K. Mitter, "Volatility of power grids under real-time pricing," *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 1926–1940, Nov. 2012.

[10] E. Ela, M. Milligan, and M. O'Malley, "A flexible power system operations simulation model for assessing wind integration," in *IEEE PES General Meeting 2011*, July 2011, 8 pp.

[11] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. B. Kopena, "Network simulations with the ns-3 simulator," presented at SIGCOMM, 2008, [Online] Abstract available: http://goo.gl/25vuwr.

[12] A. Zipperer, P. A. Aloise-Young, S. Suryanarayanan, R. Roche, L. Earle, D. Christensen, P. Bauleo, and D. Zimmerle, "Electric energy management in the smart home: Perspectives on enabling technologies and consumer behavior," *Proceedings of the IEEE*, vol. 101, no. 11, pp. 2397–2408, Nov. 2013.

[13] D. Pinney, "Costs and benefits of conservation voltage reduction," National Rural Cooperative Association, Arlington, VA, Tech. Rep., Nov. 2013. [Online]. Available: http://goo.gl/J2gnn5

[14] K. P. Schneider, Y. Chen, K. P. Chassin, R. Pratt, D. Engel, and S. Thompson, "Modern grid initiative: Distribution taxonomy final report," Pacific Northwest National Laboratory, Richland, WA, Tech. Rep., Nov. 2008. [Online]. Available: http://goo.gl/IJIAkf

[15] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *3rd International Conf. on Genetic Algorithms*, June 1989, pp. 116–121.

[16] D. C. Walters and G. B. Sheble, "Genetic algorithm solution of economic dispatch with valve point loading," *IEEE Transactions on Power Systems*, vol. 8, no. 3, pp. 1325–1332, Aug. 1993.

[17] S. A. Kazarlis, A. G. Bakirtzis, and V. Petridis, "A genetic algorithm solution to the unit commitment problem," *IEEE Transactions on Power Systems*, vol. 11, no. 1, pp. 83–92, Feb. 1996.

[18] T. Hansen, R. Roche, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel, "Heuristic optimization for an aggregator-based resource allocation in the Smart Grid," under review, Nov. 2013. Preprint available: http://www.engr.colostate.edu/sgra/.

[19] P. Palensky and D. Dietrich, "Demand side management: Demand response, intelligent energy systems, and smart loads," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, pp. 381–388, June 2011.

[20] K. Spees and L. B. Lave, "Demand response and electricity market efficiency," *The Electricity Journal*, vol. 20, no. 3, pp. 69–85, Apr. 2007.

[21] Federal Energy Regulatory Commission, "Assessment of demand response and advanced metering," Dec. 2012. [Online]. Available: http://goo.gl/gJhV2v

[22] H. Aalami, G. R. Yousefi, and M. P. Moghadam, "Demand response model considering EDRP and ToU programs," in *IEEE PES Transmission and Distribution Conference and Exposition*, Apr. 2008, 6 pp.

[23] Duke Energy Carolinas, LLC. Residential service Time-of-Use. Accessed: Nov. 15, 2013. [Online]. Available: http://goo.gl/q3K8Lt

[24] P. H. G. Berkhout, J. C. Muskens, and J. W. Velthuijsen, "Defining the rebound effect," *Energy Policy*, vol. 28, no. 6, pp. 425–432, June 2000.

[25] W. J. Cole, K. M. Powell, E. T. Hale, and T. F. Edgar, "Reduced-order residential home modeling for model predictive control," *Energy and Buildings*, vol. 74, pp. 69–77, May 2014.