OpenBuildNet Framework for Distributed Co-Simulation of Smart Energy Systems

Truong X. Nghiem, Altuğ Bitlislioğlu, Tomasz Gorecki, Faran A. Qureshi, Colin N. Jones
Automatic Control Laboratory, École Polytechnique Fédéral de Lausanne, Lausanne, Switzerland

***Abstract*—The complexity and diversity of future energy systems will require co-simulation solutions that enable the integration of tools from multiple domains for research and development. We introduce an open-source framework, OpenBuildNet, for distributed co-simulation of large-scale smart energy systems. Using a loose-coupling approach to co-simulate parallel processes, it can leverage and seamlessly integrate specialized simulation and computation tools in a common platform. Users can therefore benefit from the capabilities of state-of-the-art and widely used tools in each domain. OpenBuildNet is scalable and highly flexible as it uses a decentralized architecture, message-based communication, and peer-to-peer data exchange between subsystem nodes. It also provides a set of easy-to-use software tools tailored for researchers and engineers. This paper presents the architecture and tool suite of OpenBuildNet, and demonstrates its usefulness in a case study of controlling multiple buildings for demand response.**

I. Introduction

Energy systems are undergoing a substantial change in scale and complexity. On the grid and supply side, the embedding of distributed resources such as renewable energy sources and distributed storage systems, advanced control and optimization algorithms, communication networks, and the mass adoption of electric vehicles will all contribute to the significant increase in complexity of future grids. On the demand side, energy efficiency has always been an important goal. Future energy systems will require a tighter cooperation between the demand side and the grid, e.g., through demand side management. As the largest energy consumer in the world, the building sector is experiencing growing research and applications of advanced control and optimization techniques along these directions (see [1] for an extensive review).

While most research has focused either on a single building with little interaction with the grid or on a power grid with oversimplified demand side models, studies of the integrated system supported by large-scale, high-fidelity simulations are scarce. One reason is the lack of tools that can take on the complexity, heterogeneity, and scale of such simulations, while still ensuring the detailed and high-quality simulation of each subsystem. Specialized simulation platforms exist for each domain, e.g., MATPOWER [2] for power systems, EnergyPlus [3] for buildings, Matlab for control and optimization. However, none of them can handle the increasing interdependencies between the components of the entire system, which has led to the recent development and adoption of co-simulation in energy system research and applications. This paper presents OpenBuildNet, an open-source framework for distributed co-simulation, where specialized and sophisticated simulation tools in various fields are incorporated in a common platform to study the interdependencies between the subsystems.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 307608: BuildNet, the Swiss National Science Foundation under the GEMS project (Green Energy Management of Structures, grant n. 200021 137985), and the NRP 70 Energy Turnaround Project (Integration of Intermittent Widespread Energy Sources in Distribution Networks: Storage and Demand Response, grant n. 407040 15040/1).

A. Goals, Target Users, and Main Contributions

The main goal of OpenBuildNet is to provide a *framework and software tools for large-scale distributed co-simulation of complex systems*, with *intended applications in smart energy systems* such as smart buildings, power grids, and optimization-based controllers. Most of the time, such co-simulation systems are heterogeneous, which involves subsystems of different types and on potentially vastly different time scales. For example, a smart grid simulation may contain buildings, which have very slow thermal dynamics and fast electrical system dynamics, and the power grid with a very fast sub-second time scale. Furthermore, subsystems come in different forms and sizes: from large and highly sophisticated simulators such as EnergyPlus, to moderately complex controllers prototyped in Matlab or Python, to simple rule-based controllers implemented in C. OpenBuildNet aims to support simulation systems composed of many heterogeneous sub-simulators distributed over multiple networked computers. It allows integration of specialized or legacy tools into the co-simulation to lower the barrier for adoption and reuse their industrial-strength, highly developed functionality.

OpenBuildNet targets researchers and engineers in control, optimization, and computer science, who wish to apply their expertise and techniques to smart energy systems. The recent proliferation of optimization-based and machine learning methods particularly calls for co-simulation tools that are sophisticated enough for the intended multi-disciplinary applications, and at the same time are intuitive and accessible to these users. With OpenBuildNet, researchers and engineers can comfortably implement large-scale heterogeneous co-simulations from within their familiar scientific computing environments and languages such as Matlab and Python. They can fully utilize the functionality of these environments, including most importantly the debugging capability of a single subsystem as well as the entire co-simulation system. We found that these features are particularly important for research and development.

A main contribution of OpenBuildNet is the development of an open, scalable, and flexible framework for distributed co-simulations. Its scalability comes from the facts that the local sub-simulators are CPU-bound and loosely coupled by boundary states at discrete time instants, and that they exchange data directly with each other through peer-to-peer communications (cf. Section II). These help minimize the communication overhead and scale up the distributed computation. It also means that OpenBuildNet is highly flexible as the sub-simulators have freedom in how they communicate, therefore easing the integration of new simulation tools and the transition from simulation to real-world implementations. Another contribution is a rich suite of software tools that are tailored to the needs of our target users (see Section II-B). We have demonstrated OpenBuildNet in a case study of distributed control of multiple buildings for demand response in Section III.

B. Related Work

The growing need for large-scale and complex energy system simulation solutions has generated several co-simulation tools in recent years. In this section, we will review some of the related tools. Among the most widely used is the Building Controls Virtual Test Bed (BCVTB) [4]. It can couple simulation tools such as EnergyPlus and Matlab to a graphical modeling and simulation environment through simple master–client architecture and TCP socket communication. The simple co-simulation mechanism and communication protocol of BCVTB lower the barrier to use it, but also limit its support for complex simulation systems. In addition, the master–client approach makes it difficult to debug code and use the interactive development capability of a scientific computing environment like Matlab. Based on the same technology as BCVTB but targeting more specific applications are VirGIL [5] and SmartBuilds [6]. Using BCVTB's communication protocol, MLE+ [7] enables a better integration between EnergyPlus and Matlab, but still has most of BCVTB's limitations.

GridLAB-D [8] is a comprehensive power distribution simulation and analysis tool, which also includes building models and control algorithms. However, its co-simulation capability is limited, e.g., it does not provide an EnergyPlus interface and its Matlab interface is primitive. GridSpice [9] brings GridLAB-D to computing clusters for distributed smart grid simulations, focusing on the power distribution and transmission systems.

Originating in the automotive industry, Functional Mock-up Interface (FMI) is a standardized interface to support model exchange and co-simulation of dynamic models using XML files and compiled C code. Recently, there has been increasing support for FMI in energy system simulation platforms, e.g., in EnergyPlus and BCVTB. Although FMI does not provide any simulation platform, several simulation software packages supporting FMI are freely available [10], [11].

A common feature of the aforementioned tools is that agents exchange data through a centralized master that controls the simulation. In contrast, OpenBuildNet adopts a decentralized, peer-to-peer communication model where agents exchange data with each other directly. This distinction can increase the scalability and flexibility of OpenBuildNet as explained in Section II-A. Finally, we remark that most of the above tools, e.g., FMI and GridLAB-D, can complement OpenBuildNet and add to the toolbox that OpenBuildNet provides to researchers and engineers, as we plan to do in Section IV.

II. Overview of OpenBuildNet Framework

A. OpenBuildNet Architecture

This section briefly describes the architecture and the most important concepts of OpenBuildNet from the users' perspective. A developer who wishes to extend the framework might need to understand the low-level specifications, which are available on our website at https://sites.google.com/site/buildnetproject.

1) Nodes as the Building Blocks: A distributed co-simulation in OpenBuildNet is a *synchronous simulation distributed to multiple computation nodes*. The nodes run their own local simulations in parallel, which are synchronized and driven by a global clock at discrete time instants. A global synchronization mechanism is required because nodes, as sub-simulators, often realize dynamical systems with differing sampling rates and computation speed. This heterogeneity and multi-timescale nature is particularly typical in the target applications of OpenBuildNet in large-scale energy systems.

OpenBuildNet considers the node model illustrated in Figure 1. *Nodes exchange data with other nodes through ports.* The ports of a node represent its abstract interface to the external environment. A *physical port* is a port whose data exchanges are synchronized and strictly managed by the global clock. As a result, a physical port is either an input or an output exclusively. In contrast, a *data port* is not synchronized nor managed by the global clock and can possibly be bidirectional. Each node and port must have a unique valid identifier, which is a sequence of alphanumeric and the underscore ("_") characters and may only begin with a letter. For example, input port `i1` in Figure 1 is uniquely identified by `Node/i1`.

Because in practice a node may consist of multiple subsystems, it is functionally divided into *computation blocks* (or *blocks* for short). For instance, a building node may simulate both the fast electrical system dynamics and the slow thermal dynamics as two blocks. A block is essentially a computation unit that may read certain inputs and may compute the values of certain outputs of the node containing it. This computation is triggered either periodically at a sampling time associated with the block, or non-periodically by requests during runtime, or both. Each block must have a non-negative sampling time. The blocks of a node may have different sampling times. For example, the node illustrated in Figure 1 has two periodic blocks and a non-periodic block ($T = 0$).

2) Distributed Network of Nodes: A simulation system is therefore a network of nodes connected in a certain topology. In OpenBuildNet a system node, called the *System Management Node* (SMN), synchronizes the computation of all other nodes and drives the entire simulation. As its name suggests, it also manages the entire node network besides running the simulation, e.g., adding nodes to the network and handling

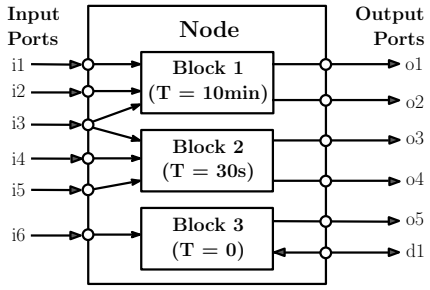

Fig. 1. Model of a node in OpenBuildNet with physical input (output) ports depicted on the left (right) and a data port. Blocks 1 and 2 are executed every 10 minutes and 30 seconds respectively, block 3 is non-periodic.

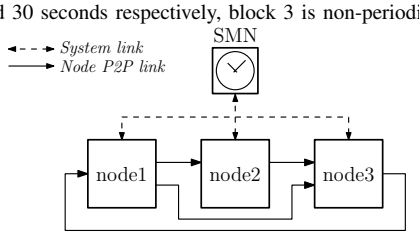

Fig. 2. OpenBuildNet network of nodes: the SMN manages the entire network and coordinates the co-simulation through system communication connections with the nodes (dashed lines). Nodes communicate directly with each other through their ports and peer-to-peer communication links (solid lines).

system errors. Every node must connect to and communicate with the SMN in order to participate in the co-simulation. This architecture is illustrated in Figure 2.

Nodes can be distributed across multiple computers over a communication network (possibly the Internet). Furthermore, nodes exchange data with each other through their ports and direct peer-to-peer links, i.e., their communications are not routed through the SMN. This decentralized architecture reduces the communication overhead and helps OpenBuildNet scale up without difficulty.

3) Execution and Synchronization: The execution of a block of a node is split into two stages. In the first stage, called *output update*, the block computes and sends out its outputs. This computation may depend directly on the current values of certain inputs; in that case the block is said to have direct feedthrough from those inputs. In the second stage, called *state update*, the block updates its internal states.

OpenBuildNet blocks are loosely coupled and only their boundary states are synchronized during the output update *stage at discrete-time steps, by exchanging data through their input and output ports.* This synchronization mechanism is controlled by the SMN via system messages. The interdependencies between blocks determined by their direct feedthrough properties and the system topology define a partial order between them at any time instant. The SMN enforces the relative order between blocks by issuing system messages in a precise order decided by a graph-based algorithm. It also detects *algebraic loops*, situations in which two blocks have circular dependency between them and their synchronization cannot be resolved. Blocks that have no relative order between them can be executed simultaneously and hence enjoy the computational speed-up benefit of distributed computation.

Details on the synchronization mechanism of OpenBuildNet, its mathematical semantics and results on its correctness can be found in the technical documents on our website.

B. OpenBuildNet Tool Suite

This section briefly describes the current implementation of OpenBuildNet, summarized in Figure 3 and explained below.

- The **communication network** is a distributed messaging system used to transport messages between all nodes. Currently, MQTT[1] and Yarp[2] are supported. On top of that is the **structured data interchange format** which encodes and decodes messages between nodes. The current implementation uses Google's Protocol Buffer[3].
- The **OBN SMN Library** implements all functionality of the SMN in C++. This library is used by the **OBN Server**, which provides a scripting language called OSS (OpenBuildNet Server Script), based on Chaiscript[4], for configuring and performing OpenBuildNet simulations.
- On the client side, the **OBN Node Library** in C++ implements all functionality of an OpenBuildNet node. However, creating a node with this library requires programming in C++ and re-compiling the code on every change, which are inconvenient considering the target users and applications of OpenBuildNet. For this reason, a collection of components were developed on top of this library to facilitate the use of OpenBuildNet.
- The **OBN External Interface** library provides a consistent interface between OpenBuildNet and an external, often high-level, language to program nodes. Packages for Matlab, Python and Julia are provided. Matlab and Python are popular among engineers and researchers - our target users - while Julia[5] is a promising new programming language for technical computing. Most importantly, these packages enable rapid and interactive development as well as convenient debugging of OpenBuildNet simulation code in these scientific environments.
- **OBNNode Scripting** (ONS) is a Chaiscript-based scripting language to program nodes, with support for common mathematical and linear algebra functions. It is lightweight and intended for creating simple nodes.
- Finally, a library of ready-made nodes commonly used in energy system simulations is provided. Currently, nodes representing buses and loads on power grids are available. In addition, a customized version of EnergyPlus called **EnergyPlus-OBN** with built-in support for OpenBuildNet was developed. It allows immediate use of any EnergyPlus model as a co-simulation node in OpenBuildNet.

The OBN External Interface provides a consistent node programming interface across multiple languages, subject to syntax differences. The workflow is always the same: a node object is created, followed by all its ports, then callback functions are defined for the output update and state update

[1] An ISO standard lightweight messaging protocol; http://mqtt.org.
[2] Yet Another Robot Platform; http://yarp.it.
[3] https://developers.google.com/protocol-buffers
[4] An embedded scripting language for C++: http://chaiscript.com
[5] http://julialang.org

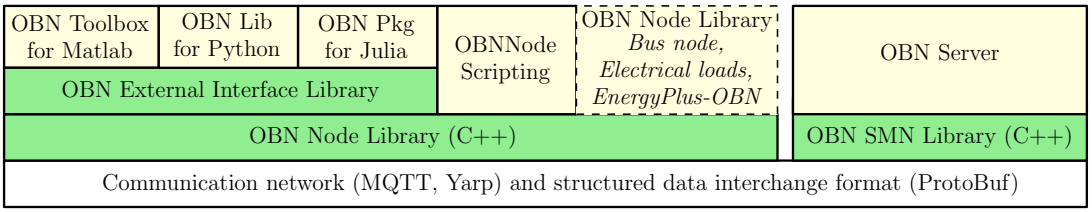

Fig. 3. The OpenBuildNet tool suite: the colored boxes represent libraries and programs developed as part of OpenBuildNet.

```python
# Create node and ports
node = OBNNode('plant')
node.x = x0
u = node.create_input('u', 'vector', 'double')
y = node.create_output('y', 'vector', 'double')
# Define and assign callbacks
def calcy():
  y.set(np.dot(C,node.x) + np.dot(D,u.get()))
def calcx(node):
  node.x = np.dot(A,node.x) + np.dot(B,u.get())
node.on_block_output(0, calcy)
node.on_block_state(0, calcx, node)
node.run()  # Run simulation
```

(a) Python node

```julia
# Create node and ports
node = OBNNode("plant")
x = x0
u = create_input(node,"u",Vector{Float64})
y = create_output(node,"y",Vector{Float64})
# Define and assign callbacks
on_block_output(node, 0) do
  set(y, C*x + D*get(u))
end
on_block_state(node, 0) do
  global x
  x = A*x + B*get(u)
end
run(node)  # Run simulation
```

(b) Julia node

Fig. 4. Implementation of the linear plant node example in different languages. A, B, C, D, and $x0$ are assumed to exist and thus not included in the code.

of each block. For example, consider a plant node described by a discrete-time linear system of the form

$$x(t+T) = Ax(t) + Bu(t);\ y(t) = Cx(t) + Du(t)$$

where x, u, y are the state, input, and output vectors respectively, and A, B, C, and D are matrices of appropriate dimensions. The self-explanatory code snippets in Figure 4 implement this node in different languages.

III. Case Study: Distributed Control of Buildings for Demand Response

This section presents a simulation study to demonstrate the capabilities of OpenBuildNet. We consider a peak-shaving demand response (DR) application in a mid-voltage distribution grid, which serves electricity to commercial buildings of various sizes. Some of these buildings are responsive to DR incentives. The simulation includes internal dynamics of the buildings, their controllers that regulate internal variables and respond to DR signals, and their interaction with the power grid, resulting in the bus voltage and branch power flow values.

The case study includes several layers of interaction between nodes, as depicted in Figure 5. The nodes in each layer operates in various time scales, such as the fast power flow evolution and slow thermal dynamics of the buildings. In the following we will summarize the structure and implementation of the case study, and some simulation results.

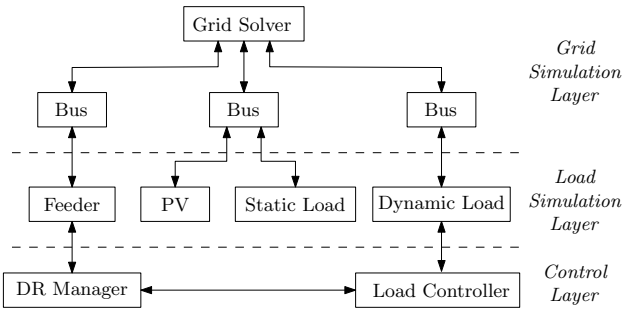

Fig. 5. Structure of the co-simulation system in the case study.

A. Simulation of the distribution grid

For the simulation of the power flow variables in an electricity distribution grid, we use MATPOWER [2], which provides a general powerflow solver for balanced networks, under steady state conditions. The simulation of the grid requires: (a) a grid node, which solves the power flow equations and determines the power injections and voltage levels; and (b) nodes representing buses in the grid, which receive inputs from the user nodes and send out the fixed power flow variables to the grid node. The user nodes may represent feeders, generators or loads and multiple users might be connected to the same bus, as shown in Figure 5.

B. Simulation of buildings

In this case study, we populate the distribution grid with commercial buildings and solar panels. The power demand of the 'static' buildings and the output of solar panels are not controlled and are pre-determined, whereas the power consumption of the 'dynamic' buildings, which are responsive to DR signals, are determined by their controllers. For the simulation of dynamic buildings, we use two different pieces of software, Matlab and EnergyPlus, to demonstrate the capability to integrate various types of software and languages.

1) Buildings with Economic Model Predictive Control: We consider a subset of buildings to be controlled by Model Predictive Control (MPC) [12].

a) Modeling: The MPC-controlled buildings are originally modeled in EnergyPlus, obtained from [13], which are too complex to use for MPC. We use OpenBuild [14] to automatically extract a linear thermodynamic model of each building from its EnergyPlus data files. OpenBuild also outputs for each building the disturbance sequences that capture solar gains, internal gains and the effect of the ambient temperature.

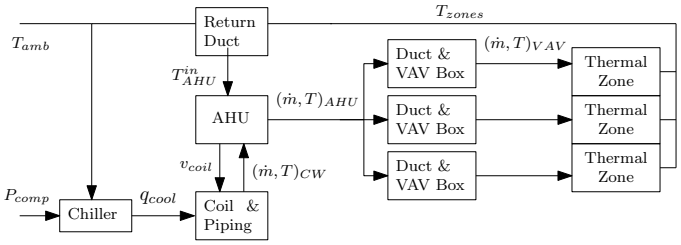

Fig. 6. Diagram of the HVAC system, which consists of a chiller, a chilled water (CW) loop, an air handling unit (AHU), duct system and variable air volume (VAV) boxes. $\dot{m}$ is mass flow rate, T temperature, P_{comp} the electrical power consumption of the chiller, q_{cool} the heat power extracted from the chilled water loop, v_{coil} the valve control parameter of the cooling coil inside the AHU. The controlled variables are $\dot{m}_{\text{VAV}}$. The input air temperature $T^{\text{in}}_{\text{AHU}}$ depends on both zone temperatures T_{zones} and ambient temperature T_{amb}, due to the mixing of internal and external air in the return duct.

As the resulting model does not capture the dynamics of the HVAC or the electrical power consumption of the building, we couple it with an extra layer of HVAC model. We choose to model a forced-air system as described in Figure 6, and size it according to the maximum heating/cooling demand of the building model. In this study we consider the cooling period. To simplify the model, we neglect the mass transport dynamics and only consider thermal dynamics of the transport media: water between the heat pump / chiller and AHU, and air in the duct system. The model is described by a continuous time nonlinear ordinary differential equation, which is solved numerically during the simulations.

b) Control: Each building is controlled by a two-level hierarchical controller. In the upper layer, an MPC controller optimizes the electrical power consumption cost over a finite horizon and specifies the thermal power input to the zones in the building. The model used in this layer captures only thermal dynamics. In order to have an estimate of the electrical energy used by the HVAC system, we use a time varying coefficient of performance (COP) estimation. For the cooling case it can be estimated as $\text{COP} = \eta \frac{T_{\text{cold}}}{(T_{\text{hot}} - T_{\text{cold}})}$, where η describes the efficiency of the heat pump. T_{cold} can be taken as equal to the water temperature at the evaporator T_{evap}, and T_{hot} can be taken as the ambient temperature T_{amb}. We use the disturbance values provided by OpenBuild for the ambient temperature prediction, whereas T_{evap} is regulated by the lower level controller at a known fixed value.

The finite horizon MPC problem can be written as

$$\begin{aligned} \min._{e_{\text{T}}} \quad & \textstyle\sum_{i=0}^{N} \pi(i) e_{\text{E}}(i) \\ \text{s.t.} \quad & \forall i, \;\; x_{\text{T}}(i+1) = A x_{\text{T}}(i) + B e_{\text{T}}(i) + D d_{\text{T}}(i) \\ & e_{\text{E}}(i) = \text{COP}(i) e_{\text{T}}(i), \;\; y_{\text{T}}(i) = C x_{\text{T}}(i) \\ & e_{\text{T}}^{\min} \leqslant e_{\text{T}}(i) \leqslant e_{\text{T}}^{\max}, \;\; y_{\text{T}}^{\min} \leqslant y_{\text{T}}(i) \leqslant y_{\text{T}}^{\max} \end{aligned}$$

where N is the horizon length, π is the electricity price. The variables x_{T}, e_{T}, d_{T} and y_{T} describes state, input, disturbance and output for the linear thermal dynamics. The input to thermal dynamics e_{T} is the thermal energy over the sampling period. The output y_{T} describes the mean zone temperatures, which are constrained to lie inside the comfort limits. The electrical energy consumption e_{E} is related to e_{T} through the forecast time varying COP.

The lower level controller consists of multiple proportional-integral (PI) controllers, which control the internal temperatures of the HVAC system as well as the thermal cooling/heating power input to the zones. See Figure 7 for details.

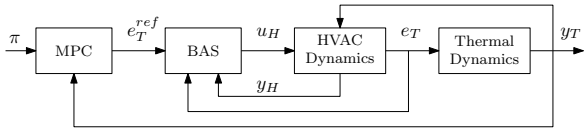

Fig. 7. Block diagram of the building control structure. The upper level MPC controller provides thermal energy e_{T} set-points to the building automation system (BAS). The BAS consists of three PI control loops that control the output of the HVAC system y_{H}, which consist of cooling power injected to the zones e_T, water temperature at the evaporator of the chiller T_{evap}, and the air supply temperature that exits the AHU, T_{AHU}.

c) Implementation: The building node has two blocks: 'Thermal' and 'HVAC'. The output of the 'HVAC' block is updated every 15 seconds by simulating the HVAC system controlled by the BAS. At each sampling of the block, the average electric power consumption and the thermal power output of the HVAC block is updated. The 'Thermal' block receives the thermal power output and simulates a linear system at a slower sampling interval of 10 minutes. Total power consumption of the node includes both the HVAC system and the uncontrollable loads in the building, the data for which is obtained from [15]. We implement large and medium sized office building nodes, and the MPC controller nodes, in Matlab. We use Yalmip [16] for parsing and Mosek [17] for solving the optimization problems.

2) Buildings with Rule Based (RB) Control: RB controllers for DR are also considered, which temporarily adjust the zone temperature setpoints of a building by 1.5 °C during the DR event to curtail its power demand. The buildings with RB controllers are simulated with the customized EnergyPlus-OBN software, while their RB controllers are implemented with OBNNode Script (ONS). Each building outputs its power demand at every 10 minutes. A variety of buildings are simulated, including two office buildings, a residential apartment building, a hospital, a hotel and a supermarket.

C. Case study

We simulate a DR application aiming to reduce the peak power injection at the feeder node. The specific grid model for the study is taken from [18] and describes a mid-voltage distribution grid with 47 buses. The implementation requires co-simulation of 42 nodes, summarized in the following table.

Node Type	Software Tools	# nodes
Matlab Building	MATLAB, OpenBuild	4
MPC Controller	MATLAB, Yalmip, Mosek	4
EnergyPlus Building	EnergyPlus	6
Rule-based Controller	OBNNode script	6
Static Building, Solar Panel	C++	14 + 5
DR Manager	MATLAB	1
Grid solver, multibus	C++	2

Apart from the nodes required for the grid and dynamic buildings, there are nodes that outputs data for static buildings, solar panels and a DR Manager which provides price forecast and DR signals. Utilizing the modular design of the OpenBuildNet framework, we distribute the nodes to three standard office computers. The study takes 36 minutes clock-time for

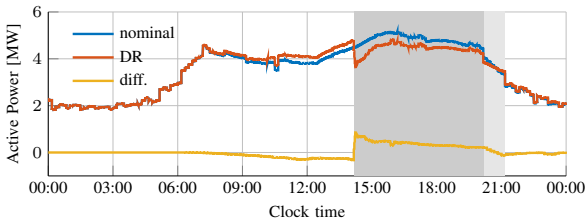

Fig. 8. Comparison of the daily active power injection at the feeder, with and without DR. The dark gray region shows the effective DR period, whereas the light gray region shows the 1-hour recovery period.

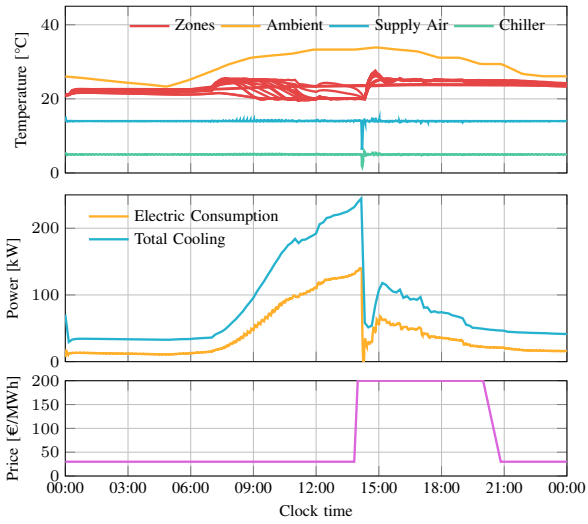

Fig. 9. Results of the DR case-study for a medium office building with 18 thermal zones. The top figure shows the evolution of mean zone temperatures, supply air temperature into the zones, and chilled water temperature at the chiller. One can observe the pre-cooling before the DR event, which reduces almost all zone temperatures to the minimum comfort constraint. In contrast, the reduced power consumption during the DR event results in zone temperatures reaching maximum comfort constraints.

24 hours simulation time, which would take several hours on a single computer. The communication between the computers is established with MQTT.

As seen from Figure 8, without DR, a peak power consumption of about 5.16 MW occurs in the late afternoon. Based on this observation we determine the DR period to be between 14:00-20:00. In order to avoid a kick back effect, there is a recovery period of one hour after the main DR event. During the event, rule-based controllers receive DR 'on' signal, whereas MPC controllers receive an increased price signal. Pre-determined electricity price throughout the day is broadcast by the DR Manager, therefore the MPC controllers apply pre-cooling before the period of increased price, as can be observed in Figure 9. Overall, the contribution of the controlled buildings results in an average active power reduction of 0.38 MW and a maximum reduction of 0.85 MW during the DR period. We omit the results concerning the powerflow variables due to space constraints in this manuscript.

IV. Conclusions

We introduced OpenBuildNet, an open-source distributed co-simulation framework for smart energy systems. It provides a rich set of software tools tailored for researchers and engineers in control, optimization and computer science, who wish to apply their methods to energy system applications. We demonstrated the usefulness of OpenBuildNet in a distributed co-simulation case study that employed multiple simulators of different types, from the high-fidelity and sophisticated building energy simulator EnergyPlus to optimization-based controllers prototyped in Matlab.

OpenBuildNet is being extended in several directions. Support for distributed co-simulations on cloud-computing services such as the Amazon Web Services is being implemented, as well as support for more programming languages and platforms. Adding interfaces with more domain-specific simulation software and other related technologies such as FMI and BACnet will complement its power and functionality. In addition, more and larger-scale case studies will be developed to demonstrate advanced methods in control, optimization and computer science in smart energy applications.

References

[1] M. Maasoumy, A. Sangiovanni-Vincentelli *et al.*, "Smart connected buildings design automation: Foundations and trends," *Foundations and Trends in Electronic Design Automation*, vol. 10, pp. 1–143, 2016.

[2] R. D. Zimmerman, C. E. Murillo-Sanchez, 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.

[3] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl *et al.*, "EnergyPlus: creating a new-generation building energy simulation program," *Energy and buildings*, vol. 33, no. 4, pp. 319–331, 2001.

[4] M. Wetter, "Co-simulation of building energy and control systems with the building controls virtual test bed," *Journal of Building Performance Simulation*, vol. 4, no. 3, pp. 185–203, 2011.

[5] S. Chatzivasileiadis, M. Bonvini, J. Matanza, R. Yin *et al.*, "Cyber-physical modeling of distributed resources for distribution system operations," *Proc. IEEE*, vol. 104, no. 4, pp. 789–806, 2016.

[6] S. Duerr, C. Ababei, and D. M. Ionel, "Smartbuilds: An energy and power simulation framework for buildings and districts," in *Energy Conversion Congress and Exposition (ECCE), 2015 IEEE*, 2015.

[7] W. Bernal, M. Behl, T. X. Nghiem, and R. Mangharam, "MLE+: a tool for integrated design and deployment of energy efficient building controls," in *ACM BuildSys'12*, 2012, pp. 123–130.

[8] D. P. Chassin, J. C. Fuller, and N. Djilali, "GridLAB-D: An agent-based simulation framework for smart grids," *Journal of Applied Math.*, 2014.

[9] K. Anderson, J. Du, A. Narayan, and A. El Gamal, "Gridspice: A distributed simulation platform for the smart grid," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2354–2363, 2014.

[10] V. Galtier, S. Vialle, C. Dad, J.-P. Tavella *et al.*, "FMI-based distributed multi-simulation with DACCOSIM," in *Symposium on Theory of Modeling & Simulation*, 2015, pp. 39–46.

[11] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, "FIDE: an FMI integrated development environment," in *Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 1759–1766.

[12] J. M. Maciejowski, *Predictive control: with constraints*. Pearson, 2002.

[13] US Department of Energy: Office of Energy Efficiency and Renewable Energy, "Commercial Reference Buildings." [Online]. Available: http://energy.gov/eere/buildings/commercial-reference-buildings

[14] T. Gorecki, F. Qureshi, and C. Jones, "Openbuild : An integrated simulation environment for building control," in *Control Applications (CCA), 2015 IEEE Conference on*, 2015, pp. 1522–1527.

[15] "OpenEI Datasets." [Online]. Available: http://en.openei.org/datasets

[16] J. Lofberg, "YALMIP : a toolbox for modeling and optimization in MATLAB," in *2004 IEEE International Symposium on Computer Aided Control Systems Design*, 2004, pp. 284–289.

[17] M. ApS, "The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28)." 2015.

[18] L. Gan, N. Li, U. Topcu, and S. H. Low, "Exact Convex Relaxation of Optimal Power Flow in Radial Networks," *IEEE Trans. Autom. Control*, vol. 60, no. 1, pp. 72–87, Jan. 2015.