Co-simulator of power and communication networks using OpenDSS and OMNeT++

Conference Paper · November 2016

DOI: 10.1109/ISGT-Asia.2016.7796538

4 authors, including:

Fernanda C L Trindade
University of Campinas

Luis(Nando) Ochoa
University of Melbourne

Co-Simulator of Power and Communication Networks Using OpenDSS and OMNeT++

Gustavo O. Troiano
University of Campinas
Campinas, Brazil
gtroiano@dsee.fee.unicamp.br

Hélder S. Ferreira
University of Campinas
Campinas, Brazil
hsf.ferreira@gmail.com

Fernanda C. L. Trindade
University of Campinas
Campinas, Brazil
fernanda@ieee.org

Luis F. Ochoa
The University of Manchester
Manchester, UK
luis_ochoa@ieee.org

Abstract—**Understanding the interactions between power and communication networks is becoming increasingly important, particularly in the context of Smart Grids. Therefore, simulation tools that allow the evaluation of these interactions will be vital in the near future. Co-simulators emerge as powerful alternatives that make it possible to integrate widely used power and communication network simulators. However, the process of building a co-simulator is a challenging task that involves knowledge from different areas such as power and energy systems, communication systems, and computer science. To support the development of advanced co-simulators and related studies, this paper presents the implementation details of a simple co-simulator that uses OpenDSS to simulate the power network, OMNeT++ to simulate the communication network, and a PHP Server and Python scripts to establish the interoperation of these software tools. The IEEE 4-bus test system integrated with a communication network is used to validate the implementation.**

Index Terms— **Co-simulation, communication networks, OMNeT++, OpenDSS, power networks**

I. INTRODUCTION

Power distribution networks are likely to experience significant infrastructural changes in order to ensure the transition towards Smart Grids. In this new environment, power distribution networks will become more complex due to the connection of a large number of new participants, such as small-to-medium scale renewables and plug-in electric vehicles, as well as new network elements, such as controllable devices, sensors and meters [1], [2]. The future management of most network participants and elements will therefore rely on a much more pervasive communication infrastructure which will allow exchanging critical monitoring and control data.

Because of the different communication needs across wide areas, Distribution Network Operators are likely to adopt a mix of communication technologies. Optic fiber, cable and Power Line Communications (PLCs) can be applied as wired communications technologies. WiMAX, LTE, Wi-Fi, ZigBee, for example, represent options of wireless technologies. However, depending on the technology, data transfers may be affected by diverse problems such as noise, latency, loss-of-packets, and, in the case of wireless communication, to topography and environmental issues (e.g. Line-of-Sight/Non-Line-of-Sight (LOS/NLOS) propagation, rain, snow, etc.) [3].

Problems with data transfers can compromise the effectiveness of future power networks. Therefore, simulation studies are required to assess the performance of the whole system composed by both the power and communication networks. In order to achieve such a goal, a suitable simulator is required. According to [2], three alternatives can be adopted to simulate power and communication networks as a single system.

The first alternative is to build a new single software tool for simulating the whole system. The main drawback of this alternative is the complexity and the fact that it is very time-consuming [2]. The second alternative is to implement software extensions for the analysis of power networks in communication network simulators or vice-versa. Nevertheless, this solution is limited to specific scenarios and might not reproduce the dynamics of these systems adequately [4], [5]. Examples of communication network extensions for power network simulators are presented in [6] to [8] and an example of a power network extension for communication network simulators is given in [9]. Finally, the third alternative, which is the main focus of this work, is to integrate a power network simulator with a communication network simulator through an independent platform. This approach is known as co-simulation and involves two or more simulators.

The process of building a co-simulator is a challenging task that involves knowledge from different areas such as power and energy systems, communication systems, and computer science. To date, however, there is limited information on the implementation of co-simulators. To support the development of advanced co-simulators and related studies, this paper presents the implementation details of a simple co-simulator that uses OpenDSS to simulate the power network, OMNeT++ to simulate the communication network, and a PHP Server and Python scripts to establish the corresponding interoperation.

The remainder of this work is organized as follows. Section II presents a review of co-simulation studies found in the literature. Section III shows the architecture of the implemented co-simulator as well as the adopted software tools and their interactions. Section IV presents details on the implementation of the co-simulator. In section V, a case study is presented to illustrate the use of the co-simulator. Finally, the main conclusions are given in section VI.

II. Review of Co-Simulation Studies

A number of simulators can be integrated to build a power and communication networks co-simulator. To simulate power networks, for example, one can use OpenDSS [10], GridLAB-D [11], Matpower [12], Matlab/Simulink [13], SimPowerSystems [14], Modelica [15], OPAL-RT [16], and RTDS [17]. To simulate communication networks, OMNeT++ [18], NS-2 [19], NS-3 [20] and OPNET [21] are among the most popular tools.

The power network simulator should be first defined according to the analysis to be carried out. For instance, OpenDSS and GridLAB-D are more suitable for power distribution networks while Matpower is mainly for power transmission networks. Nonetheless, co-simulators can be built to cater for whole power system analyses (e.g., the Framework for Network Co-Simulation integrates Matpower, GridLAB-D, and NS-3 [22]) and even advanced real-time analyses (e.g., OPAL-RT with OPNET [23], RTDS with WANem [24], [25]).

Co-simulation platforms designed for controlling the information exchange between power and communication software tools can also be used. For instance in [26], open source Mosaik [27], developed in Python, was used to integrate OMNeT++ (also open source) with power network simulators. However, Mosaik and OMNeT++ are both tools designed to drive the simulation in a co-simulation environment, and therefore bring significant implementation challenges.

In [1] a co-simulator is proposed that integrates OpenDSS (open source) and OMNeT++ using a modular software architecture based on a PHP Server and the HTTP protocol for sending the requests and responses. However, the actual implementation was done specifically for the IEEE-13 test network and Fiber-Wireless (Fi-Wi) technology.

In [3], OpenDSS and NS-2 (open source) are integrated through a platform implemented in Matlab which works as a Runtime Interface (RTI). The exchanges between Matlab (commercial) and OpenDSS are made using a Component Object Model (COM) server, and with NS-2 they are done via SSH/SCP requests.

III. Components, Philosophy and Architecture

The implementation of any co-simulator requires first understanding and defining the different interactions between the simulators and auxiliary software needed to provide interoperability. This section presents the main components, philosophy, and the adopted architecture required to build a simple, yet effective co-simulator that integrates the open source software OpenDSS and OMNeT++.

A. OpenDSS

OpenDSS is a time-driven and open source software for simulating power distribution networks that runs only on MS Windows. This well-documented simulator allows modeling several load types, photovoltaic systems, on-load tap changers, energy storage systems, and other distribution network components. It works with the COM server, allowing the user to run OpenDSS from MS Excel VBA, Matlab, Python, etc. via scripts. OpenDSS is designed with an object oriented structure, enabling new models to be added without deep modifications.

B. OMNeT++

OMNeT++ is an open source event-driven software tool. This software is developed in C++ and presents a modular architecture. OMNeT++ has a well-written and up-to-date documentation with several code examples that can be used to build the communications topology. The OMNeT++ simulation architecture is divided into the following three basic parts:

- Configuration file (.ini);
- Network Description Language (NED) file;
- C++ file.

The configuration file (.ini) is a text file used to set up the execution of the files that characterize the communication network (NED and C++ files) in OMNeT++. It also defines which communication network from an OMNeT++ project will be run in the current simulation, as more than one communication network can be modelled in the same project.

The NED files are used to describe the communication network elements and interconnections. They also create instances of the communication modules. In these files, the networks can be represented through line codes or can be graphically built.

In the C++ files, the communication events and their sequence are defined. The most basic events used in a communication node are:

- initialize(): This function is only called once when a new object of the class is created (i.e., instantiated). It schedules initial event(s) by triggering the first call(s) to handleMessage().
- handleMessage(): This function is responsible for processing the received message. If this message is addressed to the current node, this node will analyze (parse) its content. In case a new message needs to be generated, the function generateMessage() is called. Otherwise, the function forwardMessage() is called.
- forwardMessage(): This function forwards a message to another node and is only called if the node that receives the message is not its final destination.
- generateMessage(): This function is called whenever a new message needs to be created. It creates a message object and set source and destination fields.

C. Event Driven Philosophy

The integration of power and communication software tools to build a co-simulator is challenging due to the time-driven nature of the software dedicated to the analysis of power networks and the event-driven nature of the software used in the analysis of communication networks [26]. In order to overcome this problem, three approaches can be adopted [4], [5].

The first approach consists on time step synchronization: the simulators are executed independently but paused at specific times to exchange information. This process creates long time delays for the events given that the simulators have to wait fixed time steps to exchange their results – a problem known as time accumulation error [5]. A possible solution for this problem is to

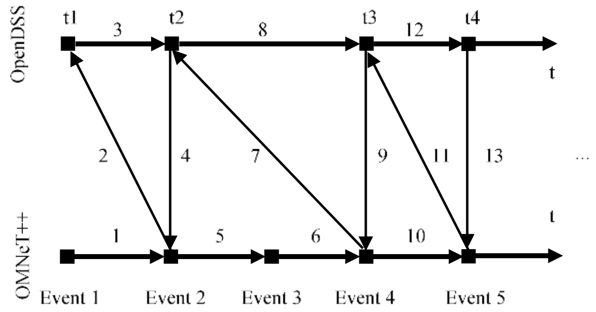

Fig. 1. Illustration of the event driven philosophy.

dynamically adjust the time synchronization [5], [28]-[30]. A second approach is to make the simulators share a list of events so as to define the times when information needs to be exchanged. In this case, when these events occur, both simulators are executed simultaneously, thus synchronizing the system. However, this approach can limit the overall co-simulation speed [4]. The third option to implement the synchronization is to use the Master/Slave approach, where the Slave is not allowed to communicate with the Master unless required. Nonetheless, one of the challenges of the Master/Slave approach is to adequately adapt the time resolution of both simulators to avoid inaccuracies when exchanging messages [2].

From the implementation perspective, one of the simplest ways to integrate OpenDSS and OMNeT++ is to adopt the Master/Slave approach, where OMNeT++ acts as the Master and OpenDSS works as the Slave. Based on this structure, selected events (i.e., not all events) are used to determine when messages will be exchanged between the individual simulators. The definition of the events and selected events depends on the study to be performed.

Whenever any of these selected events occurs, a scheduler, which is represented by a C++ class implemented inside OMNeT++, requests data from the power network. This process is illustrated in Fig. 1 where Event 3 is considered as a selected event and, therefore, triggers the exchange of messages. The scheduler plays three key roles: supervises the simulation of the communication network events; keeps track of corresponding event times; and, inserts and deletes events.

D. Architecture

The adopted architecture is presented in Fig. 2. When a selected event occurs in the communication network, OMNeT++ requests data from the power network. First, OMNeT++ checks the local Cache, which stores the last data sent by the power network. If the Cache is up-to-date, it is not necessary to run the power network simulation. Otherwise, the Communication Network Simulator Machine sends a request to Power Network Simulator Machine containing the specific time of the event. This request reaches the Controller which in turn makes OpenDSS run from the last simulation until the specified time following the load/generation behavior considered for the power network (Load Profile block). The Controller receives the data (results) from OpenDSS and sends it to the Communication Network Simulator Machine. Finally, the new data is updated in the local Cache and OMNeT++.

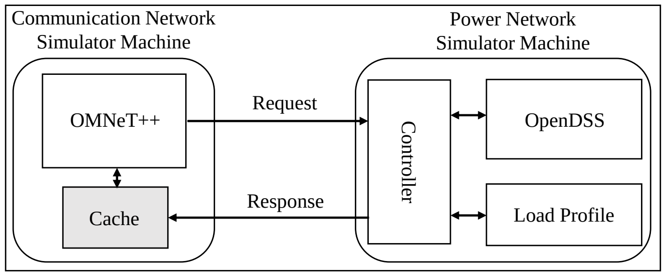

Fig. 2. Co-simulator architecture.

IV. Co-simulator Implementation

The co-simulator implemented by the authors and described in this work uses OMNeT++ version 4.6 and OpenDSS version 7.6.4.70 as the communication and power network simulators, respectively. To establish their interaction, Python 2.7.10 [31] and EasyPHP DevServer 16.1 with HTTP Apache Server 2.4.18 [32] are used. It also requires an extension package called Pywin32 Build 220 [33], to run OpenDSS from Python via the COM server.

The flowchart presented in Fig. 3 describes the whole co-simulation process. The system keeps checking whether a selected event occurred in OMNeT++. If so, the Cache is checked. The Cache validation is made through hash comparisons between OMNeT++ and Python (the name of the case study and the time of the event are checked). If the validation passes, the Cache is up-to-date. Otherwise, it is out-of-date and must be refreshed. Then, OMNeT++ sends a request to OpenDSS to update the Cache. The connection with OpenDSS is made through an HTTP POST request. With this purpose, it is possible to use “wget”, “curl” or any HTTP compliant library. To request data from OpenDSS, OMNeT++ sends a PHP Request to the Server. The PHP then extracts the time of the event and runs the Python process. For simplicity, in this implementation OpenDSS solves the snapshot (not the interval) corresponding to the load/generation level (from the Load Profile block) that matches the time of the event (or closest possible).

After the OpenDSS simulation, Python creates a JavaScript Object Notation (JSON) file (the Cache file). This file is sent to the Communication Network Simulator Machine through an HTTP response. JSON was chosen since it is compatible with several programming languages and is easily interpreted. An example of JSON file is presented in the next section.

To avoid loops and locks in the simulation execution, the number of retries for requesting data from the power network has been limited. After three unsuccessful connection attempts, the system returns an error message to the user and OMNeT++ goes back to Idle state, checking for any subsequent event on a regular basis.

This co-simulator architecture can run on the same local machine or on remote machines. The following operating systems were successfully used to run the co-simulator: Windows® (from versions 7 to 10), Linux Ubuntu® 15.10 and MAC OS X® 10.10(El Capitan).

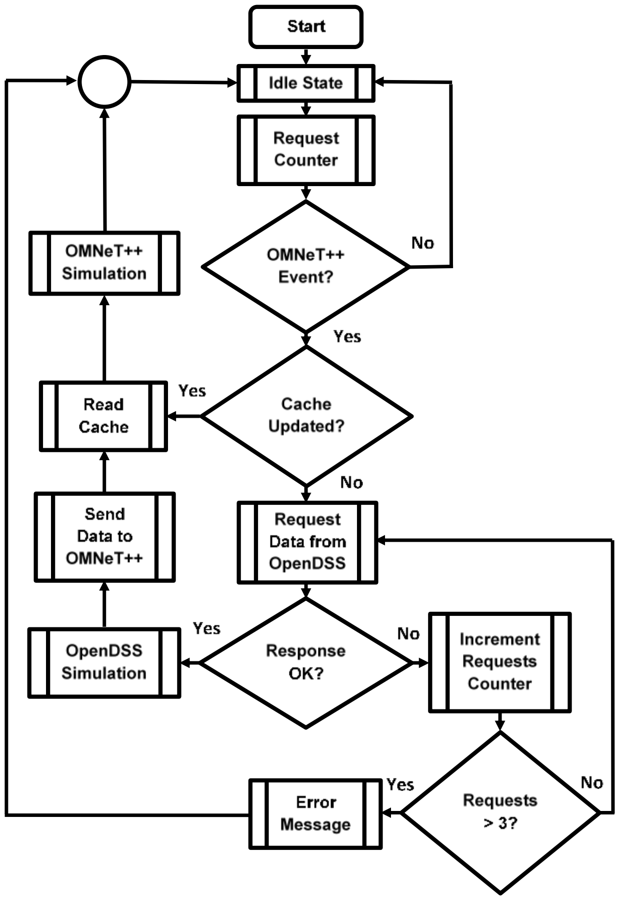

Fig. 3. Flowchart demonstrating the co-simulator functionality.

V. Case Study

The IEEE 4-bus test feeder [34], shown in Fig. 4, was used to test the co-simulator. The power network is a three-phase 4-bus network with two lines and one transformer. One line of 2,000 feet connects the source bus to the primary side of a 12.47 kV (delta):4.16 kV (wye grounded) transformer. A second line of 2,500 feet connects the secondary side of the transformer to a balanced load. This load follows a pre-defined load profile.

For this test feeder, the adopted communication network is shown in Fig. 5. It is composed of one router, one server, and four clients – each client from the communication network corresponds to a meter from the power network (same numbering). The connections between the participants of the network from Fig. 5 are described on a NED file, as follows.

```
connections:
{
  Router.gate++   <--> Backbone <--> Server.gate++;
  Client_1.gate++ <--> Wireless <--> Router.gate++;
  Client_2.gate++ <--> Wireless <--> Router.gate++;
  Client_3.gate++ <--> Wireless <--> Router.gate++;
  Client_4.gate++ <--> Wireless <--> Router.gate++;}
```

Each meter/client measures phase voltages at each of the corresponding buses of the power network. The meter/client at the source bus (bus 1 in Fig. 4) also measures the total power injected to the power network. The communication network is

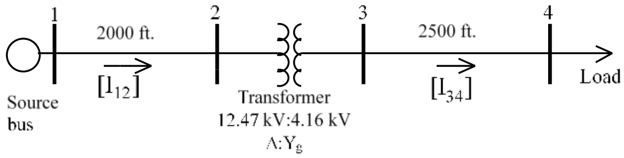

Fig. 4. IEEE 4-bus test feeder [34].

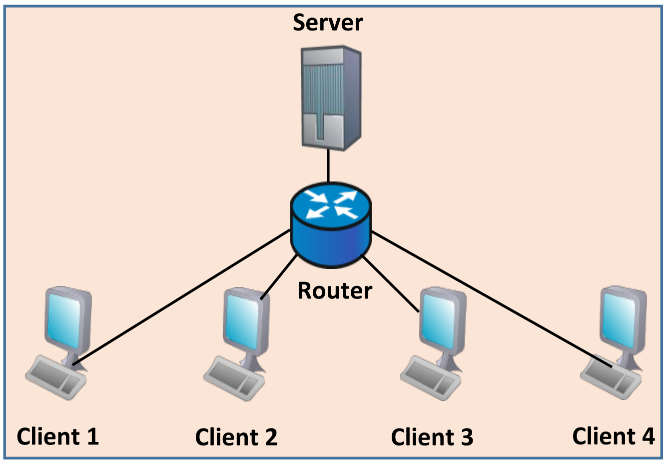

Fig. 5. Communication network integrated to the IEEE 4-bus test feeder.

modeled with a transmission rate of 54 Mbps between the clients and the router and with 1 Gbps from the router to the server. These represent typical bandwidth values found in wireless (IEEE 802.11g) and fiber technologies, respectively [35], [36]. For simplicity, no protocol was modeled and the channels are assumed to be ideal, i.e., without any Bit Error Rates (BERs) or jitters. To ensure a delay exists so OMNeT++ and OpenDSS can interact, a half-second delay is adopted for each connection of the communication network.

In the data transfer process, the clients are requested in ascending order (from 1 to 4) and the events occur as follows. The server creates and sends a request addressed to a client. Then, the router receives the message from the server and forwards it to the target client *i*, where *i* refers to one client from the set of four clients. Until the message arrives, the client is in “idle state” (see Fig. 3). When client *i* receives the message, an OMNeT++ event is detected and the Cache is checked. If the Cache is not updated, the Communication Network Simulator Machine calls the Power Network Simulator Machine to obtain the data measured at bus *i* at the current time. Once the reply from the Power Network Simulator Machine is received, client *i* sends the message containing the power network data back to the server.

Fig. 6 shows the data request from the server to client 2 in the simulation environment of OMNeT++. Fig. 7 illustrates the client 2 sending a message with the measured voltage to the server. In this simulation, all voltage values are measured in p.u. and time in seconds.

As mentioned previously, a JSON file is used to send the data from OpenDSS to OMNeT++ given its simplicity. The corresponding data is illustrated in Fig. 8 which includes values for the total power (in kW) as well as voltage magnitudes (in p.u.) and angles (in degrees) for buses 3 and 4.

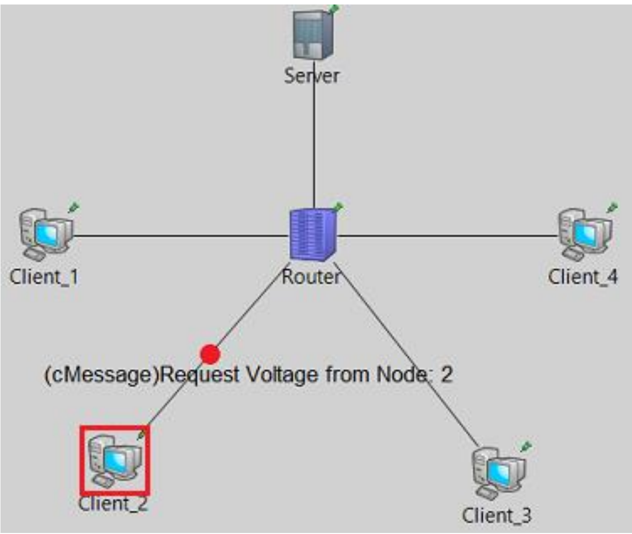

Fig. 6. Request of power network data from the Server to Meter 2 (Client_2).

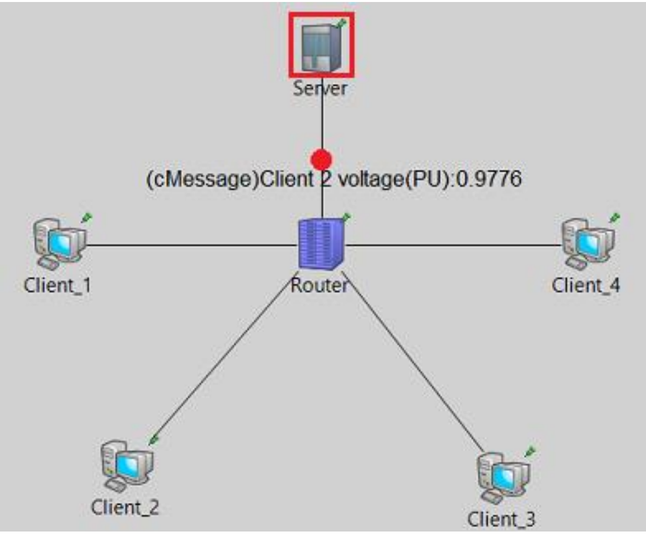

Fig. 7. Power network data sent from the Meter 2 (Client_2) to Server through the communication network.

VI. Conclusions

The simultaneous analysis of power and communication networks is likely to become a critical task in the design of future Smart Grids. To support the development of advanced co-simulators, this paper presents the main implementation details of a simple co-simulator that uses OpenDSS to simulate the power network, OMNeT++ to simulate the communication network.

Synchronization is one of the greatest challenges when implementing a co-simulator. This work used a Master/Slave approach to achieve the synchronization between the power and communication network simulators with OMNeT++ as Master and OpenDSS as Slave. The interaction of OMNeT++ and OpenDSS was established using Python scripts, and EasyPHP DevServer with HTTP Apache Server. The messages (data) from OpenDSS to OMNeT++ were built as JSON files, given their simplicity and compatibility with several programming.


```
        {
            "py/object": "meterInfo.MeterInfo",
            "v2Arg": -120.13052113134805,
            "name": "n3",
            "numberOfPhases": 3,
            "v1Arg": -0.10106209057890773,
            "v3Arg": 119.84960284043446,
            "v3Mag": 0.9951699465827121,
            "v1Mag": 0.9945488294637916,
            "powerP": 0.0,
            "powerS": 0.0,
            "v2Mag": 0.9957677887936184
        },
        {
            "py/object": "meterInfo.MeterInfo",
            "v2Arg": -120.13052113134805,
            "name": "n4",
            "numberOfPhases": 3,
            "v1Arg": -0.10106209057890773,
            "v3Arg": 119.84960284043446,
            "v3Mag": 0.9951699465827121,
            "v1Mag": 0.9945488294637916,
            "powerP": 0.0,
            "powerS": 0.0,
            "v2Mag": 0.9957677887936184
        }
    ],
    "totalLosses": 104.29843543625309,
    "numberOfBuses": 4,
}
```


Fig. 8. A JSON picture sample representing part of the power system parameters.

The implemented co-simulator was successfully validated with the integrated analysis of the IEEE 4-bus test feeder and a simple communication network.

VII. References

[1] M. Lévesque, M. Maier, D. Q. Xu., G. Joós, "Communications and Power Distribution Network Co-Simulation for Multidisciplinary Smart Grid Experimentations," in *Proceedings of the 45th Annual Simulation Symposium (ANSS'12)*, Orlando, FL, 2012.

[2] B. Amarasekara, C. Ranaweera, A. Nirmalathas, R. Evans, "Co-simulation platform for Smart Grid applications," in *IEEE Innovative Smart Grid Technologies - Asia (ISGT)*, Bangkok, Thailand, Nov. 2015.

[3] M. Garau, G. Celli, E. Ghiani, G. G. Soma, F. Pilo, S. Corti, "ICT reliability modelling in co-simulation of smart distribution networks," in *IEEE 1st International Forum on Research and Technologies for Society and Industry (RTSI) Leveraging a better tomorrow*, Turin, Italy, Sept. 2015.

[4] W. Li, M. Ferdowsi, M. Stevic, A. Monti, F. Ponci, "Cosimulation for Smart Grid Communications," *IEEE Trans. Industrial Informatics*, vol. 10, no. 4, pp. 2374-2384, Nov. 2014.

[5] A. T. Al-Hammouri, "A comprehensive co-simulation platform for cyber-physical systems," *Computer Communications*, vol. 36, no. 1, pp. 8-19, Dec. 2012.

[6] TrueTime: Simulation of Networked and Embedded Control Systems. [Online]. Available: http://www.control.lth.se/truetime/.

[7] Prowler: Probabilistic Wireless Network Simulator. [Online]. Available: http://www.isis.vanderbilt.edu/projects/nest/prowler/.

[8] VisualSense. [Online]. Available: http://ptolemy.eecs.berkeley.edu/visualsense/

[9] NRL's Sensor Network Extension to NS-2. [Online]. Available: http://www.nrl.navy.mil/itd/ncs/products/sensorsim.

[10] OpenDSS EPRI Distribution System Simulator. [Online]. Available: https://sourceforge.net/projects/electricdss/files/OpenDSS/

[11] U.S. Department of Energy GridLAB-D. [Online]. Available: http://www.gridlabd.org/

Accepted Paper

[12] Matpower – A MATLAB Power System Simulation Package. [Online]. Available: http://www.pserc.cornell.edu//matpower/.

[13] MathWorks Simulink. [Online]. Available: http://www.mathworks.com/products/simulink/

[14] MathWorks SimPowerSystems. [Online]. Available: http://www.mathworks.com/products/simpower/

[15] OpenModelica. [Online]. Available: https://www.openmodelica.org/

[16] OPAL-RT Technologies. [Online]. Available: http://www.opal-rt.com/

[17] RTDS Technologies. [Online]. Available: https://www.rtds.com/

[18] OMNeT++ Discrete Event Simulator. [Online]. Available: https://omnetpp.org/

[19] NS-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[20] NS-3. [Online]. Available: https://www.nsnam.org/

[21] Riverbed OPNET. [Online]. Available: http://www.riverbed.com/products/steelcentral/opnet.html?redirect=opnet.

[22] Pacific Northwest National Laboratory, "GridOPTICS™ Open-Source Tools," [Online]. Available: http://gridoptics.pnnl.gov/articles/o/p/e/Open-Source_Tools_fa67.html

[23] D. Babazadeh, M. Chenine, K. Zhu, L. Nordstrom, A. Al-Hammouri, "A platform for Wide Area Monitoring and Control System ICT analysis and development," in *IEEE PowerTech Grenoble* (PowerTech 2013), Grenobla, France, Jun. 2013.

[24] WANem – The Wide Area Network emulator. [Online]. Available: http://wanem.sourceforge.net/.

[25] A. Benigni, A. Monti, "Development of a platform for hardware in the loop testing of network controller," in *Proc. Summer Simul. Multi-Conf. Grand Challenges Model. Simul.* (GCMS), pp. 124–128, Netherlands, 2011.

[26] J. Dede, K. Kuladinithi, A. Foster, O. Nannen, S. Lehnhoff, "OMNeT++ and mosaik: enabling simulation of Smart Grid communications," in *2nd OMNeT++ Community Summit, IBM Research*, Zurich, Switzerland, Sep. 2015.

[27] Mosaik. [Online]. Available: https://mosaik.offis.de/

[28] H. Georg, S. C. Muller, N. Dorsch, C. Rehtanz, C. Wietfeld, "INSPIRE: Integrated Co-Simulation of Power and ICT Systems for Real-Time Evaluation," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Vancouver, BC, Canada, Oct. 2013.

[29] S. Ciraci, J. Daily, K. Agarwal, J. Fuller, L. Marinovici, A. Fisher, "Synchronization algorithms for co-simulation of power grid and communication networks," in *IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, Paris, France, Sept. 2014.

[30] J. C. Fuller, S. Ciraci, J. A. Daily, A. R. Fisher, M. Hauer, "Communication simulations for power system applications," in *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, Berkeley, CA, USA, May. 2013

[31] Python™. [Online]. Available: https://www.python.org/

[32] EasyPHP. [Online]. Available: http://www.easyphp.org/.

[33] Pywin32 Python for Windows Extensions. [Online]. Available: https://sourceforge.net/projects/pywin32/files/pywin32/Build%20220/

[34] IEEE Power & Energy Society, "Distribution Test Feeders," [Online]. Available: http://ewh.ieee.org/soc/pes/dsacom/testfeeders/

[35] Q. D. Ho, Y. Gao, G. Rajalingham, T. L. Ngoc, Wireless Communications Networks for the Smart Grid, Montreal, QC, Canada: CRC Press, 2003.

[36] H. J. A. Ferrer and E. O. Schweitzer, III, Modern Solutions for Protection, Control, and Monitoring of Electric Power Systems, Pullman, WA: Schweitzer Engineering Laboratories, Inc., 2010.

Accepted Paper