Design of the HELICS High-Performance Transmission-Distribution-Communication-Market Co-Simulation Framework

Preprint

Bryan Palmintier and Dheepak Krishnamurthy
National Renewable Energy Laboratory

Philip Top and Steve Smith
Lawrence Livermore National Laboratories

Jeff Daily and Jason Fuller
Pacific Northwest National Laboratories

Presented at the 2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems
Pittsburgh, Pennsylvania
April 21, 2017

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Conference Paper
NREL/CP-5D00-67928
September 2017

Contract No. DE-AC36-08GO28308

NOTICE

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Available electronically at SciTech Connect http:/www.osti.gov/scitech

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
OSTI http://www.osti.gov
Phone: 865.576.8401
Fax: 865.576.5728
Email: reports@osti.gov

Available for sale to the public, in paper, from:

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312
NTIS http://www.ntis.gov
Phone: 800.553.6847 or 703.605.6000
Fax: 703.605.6900
Email: orders@ntis.gov

Cover Photos by Dennis Schroeder: (left to right) NREL 26173, NREL 18302, NREL 19758, NREL 29642, NREL 19795.

NREL prints on paper that contains recycled content.

Design of the HELICS High-Performance Transmission-Distribution-Communication-Market Co-Simulation Framework

Bryan Palmintier and Dheepak Krishnamurthy
National Renewable Energy Laboratory
Golden, CO

Philip Top and Steve Smith
Lawrence Livermore National Laboratories
Livermore, CA

Jeff Daily and Jason Fuller
Pacific Northwest National Laboratories
Richland, WA

***Abstract*—This paper describes the design rationale for the Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS), a new open-source, cyber-physical-energy co-simulation framework for electric power systems. HELICS is designed to support very-large-scale (100,000+ federates) co-simulations with off-the-shelf power-system, communication, market, and end-use tools. Other key features include cross-platform operating system support, the integration of both event-driven (e.g., packetized communication) and time-series (e.g., power flow) simulations, and the ability to co-iterate among federates to ensure physical model convergence at each time step. After describing the requirements, we evaluate existing co-simulation frameworks, including High-Level Architecture (HLA) and Functional Mockup Interface (FMI), and we conclude that none provide the required features. Then we describe the design for the new, layered HELICS architecture.**

Keywords—co-simulation; cyber-physical; power systems modeling; integrated transmission-distribution simulation; information and communication technologies

I. Introduction

Energy systems and their associated information and communication technology (ICT) systems are becoming increasingly intertwined. As a result, effectively designing, analyzing, and implementing modern energy systems increasingly relies on advanced modeling that simultaneously captures both the cyber and physical domains in combined simulations [1].

For the electric power system, the rapid growth in distributed energy resources (DERs)—such as solar photovoltaic (PV), storage, electric vehicles, and responsive demand—makes it increasingly critical to simultaneously capture both the transmission and distribution systems. The modernization and expansion of smart-grid ICT infrastructure introduces further interdependencies that are also necessary to capture. In addition, increasing interest in bringing market approaches to the distribution-system level suggests a need to consider transmission and distribution physical phenomena, communication infrastructure, and bulk and distributed market (TDC+M) interactions.

For example, past research has shown that accurately capturing communication system delays reveals multifold increases in simulated transmission system voltage recovery time with some proposed wide-area control architectures that are not captured by traditional analysis that ignores communication effects [2]. Similarly, communication system delays can introduce price spikes in distribution-level transactive energy markets [3], a phenomena that requires simultaneously capturing full TDC+M interactions.

Although some customized stand-alone tools (e.g., [4]–[6]) can capture a subset of these cyber-physical phenomena for reduced-scale systems, fully capturing their interactions has increasingly relied on co-simulation (e.g., [2], [7]–[10]); however, as will be discussed in Section III, current co-simulation approaches face a number challenges, particularly when fully converging among federates and when scaling to the size and complexity of realistic electric grids.

This paper describes the design of the Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS), a new, layered, high-performance co-simulation framework that builds on the collective experience of multiple national laboratories [2], [9], [10] to offer increased scalability and advanced features for modeling highly integrated cyber-physical-energy systems. Section II introduces the requirements for this system, and Section III presents an evaluation of existing co-simulation frameworks. Section IV presents an in-depth description of the layered HELICS architecture. Section V concludes and discusses next steps.

This work was supported by the Grid Modernization Laboratory Consortium and conducted by the National Renewable Energy Laboratory, Pacific Northwest National Laboratory, and Lawrence Livermore National Laboratory, which are supported by the U.S. Department of Energy (DOE) under Contracts No. DOE-AC36-08GO28308; DE-AC05-76RL01830; and DE-AC52-07NA27344 respectively.

II. Establishing Requirements

As part of the U.S. Department of Energy's Grid Modernization Initiative [11], the team first developed a comprehensive set of grid analysis test cases, including:

- Impact of DER's on bulk system reliability;
- Load modeling under high penetrations of DERs;
- Wide-area voltage stability support using DERs;
- Voltage and frequency ride-through settings for smart inverters;
- Real-time co-simulation of electric power systems and communication networks;
- Communication architecture evaluation for high penetrations of solar;
- Centralized vs. distributed control paradigms to prevent voltage stability collapse;
- Wide-area monitoring, protection, and control;
- Impacts of DERs on wholesale market operations;
- Mitigating transmission-distribution interface congestion through demand-side management;
- Regional coordinated electric vehicle charging; and
- Real-time coordination of large-scale PV and energy storage.

The HELICs team then evaluated these use cases to determine the key requirements for the co-simulation framework:

- **Highly scalable:** able to support the co-simulation of 2–100,000+ federates, thus supporting the entire range from small-scale interaction studies to interconnection-scale use cases;[1]
- **Cross-platform:** supporting everything from Windows/OSX laptops to Linux high-performance computers to enable both use of commercial tools and very-large scale analysis;
- **Modular:** able to easily build-up co-simulations from a wide variety of tools in a number of arbitrary scenarios;
- **Minimally invasive:** able to rapidly integrate diverse, existing, domain-specific simulation tools, without requiring extensive interface development and without expecting roll-back support;
- **Open source**, but able to interact with commercial power-systems tools;
- Able to support **a wide range of simulation types:**
  - Discrete-event simulation (e.g., communication)
  - Quasi-steady-state time series (e.g., time-varying power flow)
  - Phasor dynamics (e.g., transient stability analysis); and
- **Capable of co-iteration** to enable inter-federate convergence before advancing time, such as to ensure transmission-distribution power flow convergence.

[1] The number 100,000 is the estimated number of federates required to capture the transmission system, sufficient distribution feeders, multiple balancing authority area markets, control systems, and communication interactions for the Western Interconnection.

III. Evaluating Existing Co-simulation Frameworks

Before embarking on a new design, we evaluated multiple existing co-simulation frameworks to determine if an off-the-shelf solution met the requirements and to learn best practices and existing innovations to incorporate into HELICS.

A. Functional Mockup Interface

Functional Mock-up Interface (FMI) is a tool-independent standard to support dynamic differential equation-based model exchange and more general co-simulation [12] The standard presents a well-designed API for electrical, mechanical, and software simulation, allowing tools that have FMI capabilities to integrate, simulate, and analyze the coupling of models. FMI is ideal for simulating dynamic, detailed system models because the interface can efficiently describe the detailed the equations of such models and because many off-the-shelf tools support such model exchange. The co-simulation interface provides a standard for coupling multiple full simulation tools rather than equation-based models.

For co-simulation, FMI uses a master-slave arrangement with the master being directly analogous to the desired co-simulation hub. Though there are multiple slave-ready models and tools for slave generation, the set of masters is somewhat more limited. The majority of FMI master implementations are specialized for particular purposes, such as automotive simulation or whole building modeling, and the available generic co-simulation masters are typically either designed to work with functional mock-up units (FMUs) generated by a specific tool and/or are only available for a single platform, e.g. Windows. Further, many generic masters would be difficult to deploy in an HPC environment because they are Java-based.

We did evaluate the use of a Python-based FMI co-simulation master, but quickly realized it would not scale to the desired number of federates or provide the desired high-speed runtime performance. It also would be difficult to support co-iteration (multiple iterations for convergence within a single time step), particularly with existing interfaces. Still, although FMI did not provide an off-the-shelf solution for the TDC+M co-simulation needs, it did provide a number of useful design inspirations including its lightweight interface description, and the value based data transfer mechanisms are heavily influenced by the design of the FMI standard. As a result, HELICS plans to directly support FMUs, thereby providing access to a range of existing models and controls.

B. High-Level Architecture

High-Level Architecture (HLA) is a U.S. Department of Defense specification for co-simulation [13]–[15]. HLA targets distributed simulations with support for multiple programming languages and platforms. In HLA, the run-time infrastructure (RTI) provides time synchronization and communication among the federated simulators. Multiple implementations exist, including several open-source versions.

A principle concern was the ability of the HLA RTI to scale to the size of problems of interest (100,000 federates). Previous research highlighted performance concerns with HLA and proposed innovative high-performance HLA RTI implementations, but these do not appear to be production ready [16], [17]. Of the well-supported open-source HLA

implementations, previous studies showed that the CERTI HLA-based Ptolemy framework scaled reasonably well for thousands of actors [18]. However, the CERTI HLA RTI uses a centralized process (called the RTIG) to which all federates connect when registering [19], [20]. The connection to the RTIG is managed through a separate process (RTIA) that is launched during federate initialization; each launched federate thus has two processes: the federate and an RTIA. The centralized architecture using a single RTIG process raises concerns about very large scaling.

To evaluate larger-scale simulations, we performed additional scaling tests on the CERTI HLA RTI directly. We expanded a simple process/controller example program from the CERTI HLA RTI distribution to support multiple client federates interacting with one controller federate. For our indented problem space, the controller federate approximates a transmission system, and the client federate approximates a distribution system. Our extensions supported an arbitrary number of distribution systems. Data was exchanged at 1 Hz for 300 seconds resulting in 300 message exchanges per simulation. Though an extremely limited test, it enabled a quick evaluation of how well the CERTI RTI RTIG could launch and run a large number of federates. The study used Lawrence Livermore National Laboratory's Cab system, a standard Linux cluster machine with 1,200 batch nodes each with 2 Intel 8-core Xeon E5-2670 computer processing units and 32 GB of memory. Nodes are interconnected with a high-speed InfiniBand network.

We evaluated "weak" scaling—how performance changes when computing resources are increased proportionally to problem size. This scaling is important for many TDC+M studies of interest because with off-the-shelf tools, the amount of available memory can drive a need to scale to large numbers of nodes. For example, past integrated transmission-distribution simulations have found that typical GridLAB-D distribution models use approximately 1 GB of memory each [21]. The available large-scale (1,000 or more nodes) systems at the U.S. Department of Energy national laboratories have 24–64 GB per node, with 12–32 cores per node. Using these observations as a guide, we chose to launch 64 distribution stand-in processes per node and scale to the number of nodes used; the transmission and RTGI processes where run on a separate node. Fig. 1 shows the total run time for the simulation with 64–1,024 distribution systems (2–17 compute nodes). The runs do not scale well; ideally, the time should be constant. At larger scales, we observed failures to launch because federates attempted to contact the single RTGI process. Extrapolating the curve shows that running 100,000 federates would be infeasible.

In addition to scalability, we were interested in portability across platforms. To evaluate portability beyond X86 cluster supercomputers, we attempted to build CERTI HLA on an IBM BlueGene architecture, but had to abandon the effort because of the number of dependencies required by CERTI HLA RTI and the cross-compiling environment on the IBM BlueGene machine. We also found a compilation bug in the implementation for big endian architectures indicating that CERTI is not routinely compiled/tested on PowerPC machines. Portability to architectures other than X86 clusters is interesting because several of the new exascale architectures are not based on X86.

Despite these shortcomings, HLA has many powerful design features that we are adopting in HELICS.. The HLA's community's extensive work on time synchronization [22] is heavily influencing the time-advancement strategies in HELICS, and. we are employing the conservative time-synchronization approach adopted by HLA. The use of a callback-based API is also being leveraged to help avoid polling and busy loops.

C. *Framework for Network Co-Simulation*

The Framework for Network Co-simulation (FNCS) [10] is a lightweight co-simulation platform implemented in C++ with interfaces for C, Java, MATLAB, and Python. The FNCS API features an intentionally small set of functions for data exchange and time synchronization. Although FNCS is not an implementation of HLA, it borrows the ideas from a publish-and-subscribe interface and requests time advances. All data exchanges and time requests are made to a centralized broker application running as a separate process. Inter-process communication uses the flexible ZeroMQ library [23].

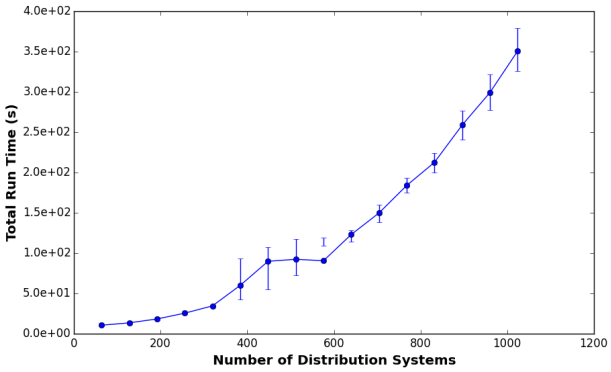

Fig. 1. "Weak" scaling study on 2-17 compute nodes with 64 federates per client node shows poor scalability because time is nearly proportional to the number of distribution systems rather than remaining nearly constant.

FNCS has been tested as part of a domain simulation [24] as well as stress-tested for this work. The domain test consisted of a transactive control layered over integrated transmission and distribution systems. The case studies involved 200 distribution systems modeled by one instance of GridLAB-D [6] each. Although the case studies were relatively small, the scaling study in [24] showed that FNCS also scales well for a few thousand federates. For this paper, we further explored FNCS support for tens of thousands of simple federates, but the performance was too slow. Further, these simple federates did not perform any real simulation and subscribed to two data streams each, so we expect that a real domain simulation of this size would perform even worse.

In addition to scalability concerns, FNCS is missing some key capabilities needed for our complete set of use cases, notably FNCS does not support co-iteration within a time step to guarantee convergence among federates. The streamlined interface presented by FNCS, though useful for rapid prototyping, also uses character strings for all communication,

presenting a large overhead and conversion performance challenges. Given the scalability issues, as well as its limited interface and capabilities, we opted not to use FNCS as the basis for HELICS.

IV. Framework Design

Based on this evaluation, the team determined that existing co-simulation frameworks do not meet all of the identified requirements, so we embarked on the design of HELICS.

A. A Layered Approach

To optimize performance; speed development; and enable clean, modular maintainability, HELICS utilizes a layered architecture (Fig. 2). Clear Application Programming Interfaces (APIs) between each layer, enable development of the individual layers to occur in parallel, with each layer free to make internal changes and optimize performance without impacting the other layers. The rest of this section describes these layers in more detail from the bottom up.

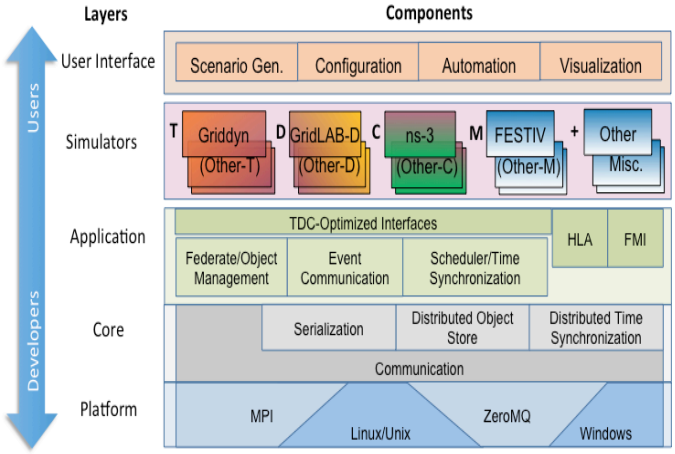

Fig. 2. The layered co-simulation architecture showing key subcomponents in each layer.

B. Platform

Careful thought is required at this lowest level to ensure that HELICS can work across operating systems—including Windows, Linux, and Macs—and across computational scales, from small laptops to large high-performance computer (HPC) clusters. To support a wide variety of platforms, two distinct communication interfaces are supported. On HPC systems with a low latency interconnect, the commonly used Message Passing Interface (MPI) is supported for inter-federate communication; on other systems where MPI is not available or necessary, ZeroMQ will be used. The choice will be made when the core and associated communication API is instantiated.

The core and API-related software are also being written in cross-platform C++, using C++14 features. A C-based interface will be available as a shared library for applications that either cannot use C++ or access a sufficient compiler. The build system itself uses CMake, and it is tested regularly on multiple platforms. The structure of the platform itself is designed to integrate easily with existing packages and build on experience with previous generations of co-simulation tools.

C. Core Layer

The core layer represents the minimum set of features necessary for a co-simulation, including time management and data exchange for combined discrete event (e.g. communication) and time series (e.g. power system) simulations. These features directly influence the run-time mechanics and cannot be abstracted to a higher level. The core layer represents an abstract interface that will be implemented by both a ZeroMQ- and MPI-based backend, giving maximum performance for desktops, small clusters, and the high-performance networks of supercomputers.

The core layer contains constructs to model value based and message based interactions from defined end points. In addition to end-point registration and explicit pairwise communication among end points, an end-point filter can be attached to orchestrate special operations, such as communication latency or complex message interactions. By exposing the semantics of end-point communication, our core layer is ready to explicitly model network communication within the application layer.

The core layer features semantics similar to other co-simulation standards. After initializing the core layer, federates are registered to the federation. Multiple federates can be registered within the same process, taking advantage of modern multithreading constructs. Time management is a key capability, allowing federates to operate with differing timescales as well as co-iterate at any time step to achieve convergence. Time management is coordinated via 2 global synchronization calls to enter initialization and execution states, and a timeRequest function to indicate to the core that the federate is ready to proceed to the next time step or that it requires further co-iteration exchange The core coordinates among the different federates to determine which federates can proceed in an ordered manner. Application Layer

The application layer is the primary low-level interface among application federates interacting with the co-simulation framework and the core API. Although the core API communication layer was designed to be simple and generic, the application layer API is intended to make it easier for generic applications of different types to interact in a flexible fashion. The API defines three types of federates with different types of interaction supported by the core.

1) Value Federates: Value federates interact through a publish-and-subscribe mechanism: outputs are published, and inputs are subscribed. The actual nature of the values is arbitrary and includes explicit support for both generic data blocks and many common types, such as floating-point numbers, integers, strings, complex numbers, and arrays. These types can be checked for matching types and will support unit conversion. The federate API also provides functions to query if a value is updated, obtain the value, and note the time of the update. Classes are also available that encapsulate the interactions of a single subscription or publication. The structure and definitions for a value federate are intended to match the features of an FMU for co-simulation, and an FMI-specific application will be made available to directly support co-simulation FMUs. Value federates also support iterative loops or superdense time steps in the FMU nomenclature.

2) Message Federates: While the value federate is targeted at applications and components interacting at a direct physical level, the message federate is intended to interact with federates simulating an ICT exchange. Unlike value publication in a value federate, which has no specific destination; a message has a specific source, destination, and time sent. These messages could represent communication packets, events, or anything else that two federates mutually understand. A message federate defines end points that are the sources and destinations for the message-based federate interaction. An end point may also subscribe to a value-based publication.

3) Message Filter Federates: The third type of federate arose from a requirement to support communication simulations at various levels without forcing the other message federates to be aware of them. This concept requires the ability of a federate to insert itself into the interaction paths. The message filter federate builds on a message federate to add additional ability to modify or manipulate a message itself or its timing. Message filters can be defined for sources and/or destinations. For example, consider modeling the interaction of an automatic generation control (AGC) system with a generator. In the simple model, control signals are sent as messages from the AGC controller directly to the generator. A more complex co-simulation requires that the full communication path between the generator and controller be modeled. With the message filter API, filters can be inserted to translate the orginal message to a specific communication packet format, to send the packet through a full communication network simulation, and to decode the packet back to the raw signal the generator model itself understands, all without changing anything in the generator or controller federates. The message structure itself is such that it keeps a record of the original source and destination end points as well as the most recent intermediate end point. This structure allows for things such as message delays, random loss, message translation, or full-stack communication simulation to be included in a co-simulation without requiring existing federates to be aware of the individual filter manipulations.

4) Programming Interfaces: A number of programming options will be available to allow user flexibility to fit with a variety of existing simulators. The primary interface is developed in C++, and a C++ API will be available making use of C++14 standards. A C shared library will be available to support applications requiring a simpler, C-style interface and for alternate compiler support. In addition, MATLAB and Python API's will also be available. The API package will also include a number of supporting classes that can directly translate numerical values or other objects in C++ to published values or messages and that otherwise attempt to make low-level interactions with the co-simulation framework as easy as possible.

5) Interaction with Existing Co-simulation Standards: Although the underlying core will not interact directly with existing co-simulation standards such as FMI and HLA, the application layer will expose interfaces for these standards. Since, the designs and experience with those standards had a significant influence on the structure and design of HELICS, many functions and features from these standards map directly to concepts and functions in the core and application layers. Many FMUs for co-simulations functions will map to the concepts in the value federate; and through the FMU-specific application, the co-simulation framework will be able to act as an FMU master, seamlessly supporting the interaction of FMU co-simulation objects with other tools not based on FMU. The full HLA architecture is much richer, but we also intend to introduce HLA-oriented support for corresponding features.

6) Interface Flexibility and Local Routing: The internal design of HELICS allows for intermediaries between different layers. An application can opt to include references to the core HELICS libraries directly ; or it can connect through an application API layer, which could include an independently executing application wrapped around the core API. This flexibility in the framework is designed to improve the scaling of the application as a whole, and provide sufficient flexibility to adapt to different types of co-simulation.

D. Simulators Layer

Although the co-simulation environment will support any federate that meets the minimum requirements, it is envisioned that many TDC+M applications will rely heavily on a common style of simulators. As a result, HELICS defines two classes of simulator interfaces:

1. General purpose, including HLA and FMI co-simulation interface support for arbitrary user-provided federates such as custom controllers; and
2. Optimized interfaces for common TDC+M application types:
   - Transmission simulator (e.g., GridDyn, PSS/E),
   - Distribution simulator (e.g., GridLAB-D, OpenDSS, CYMEDIST),
   - Communication simulator (e.g., simple delays, ns-3), and
   - Market simulator (e.g., Flexible Energy Scheduling Tool for Integrating Variable Generation, or FESTIV [25]).

To facilitate these common use cases, the simulator layer provides two key extensions: standardized data exchange patterns (variable naming, types, timing/synchronization, etc.) for these simulators in common configurations, and a higher-level API for certain common TDC+M co-simulation operations (e.g., three-phase to positive-sequence voltage send/receive).

E. User Interface Layer

Often with integrated co-simulations, particularly at medium to large scales, the simulation itself proves to be the easy part, with considerable more effort required to assemble all of the required input data, organize and run the co-simulation, and parse the results [21]. The HELICS user interface layer attempts to streamline these processes by providing tools and standardized approaches to:

- Manage and convert input data;
- Structure folders;
- Generate scenarios and populate required data;
- Automate simulation runs;

- Display simulation status and progress; and
- Collect, visualize, and compare results.

The user interface layer builds on similar past efforts among the team including [9], [26].

V. Conclusions and Next Steps

Next-generation analysis of modern electric power systems increasingly requires capturing TDC+M interactions; however, despite past research successes, no known existing environment is fully capable of such analysis, particularly at a very large scale and across multiple-platforms.

As described above, the new HELICS framework will allow for unprecedented ability to simulate modern power systems. Through co-simulation, it will enable leveraging existing off-the-shelf tools for TDC+M, user defined controls and other models. Its layered architecture enables both high-performance simuations and efficient software development. It also makes it possible to run HELICS on a wide range of platforms, from Windows/OSX laptops to Linux-based HPCs.

With the initial open-source release of HELICS (scheduled for mid-2017), the key components for each layer will be in place, along with links to a core set of simulators enabling a number of next steps. Futher development priorities include documentation, integrating advanced features, evaluating the scalability of the tool, developing interfaces for additional simulators, and establishing an active community of contributors. This initial version of HELICS will also enable novel research and analysis, including evaluating a range of use cases and further demonstrating and characterizing the value of integrated TDC+M analysis.

References

[1] P. Palensky, E. Widl, and A. Elsheikh, "Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods," IEEE Trans. Syst. Man Cybern. Syst., vol. 44, no. 3, pp. 318–326, Mar. 2014.

[2] B. M. Kelley, P. Top, S. G. Smith, C. S. Woodward, and L. Min, "A federated simulation toolkit for electric power grid and communication network co-simulation," in 2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015, pp. 1–6.

[3] J. C. Fuller, S. Ciraci, J. A. Daily, A. R. Fisher, and M. Hauer, "Communication simulations for power system applications," in 2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013, pp. 1–6.

[4] P. Evans and New Power Technologies, "Verification of Energynet® Methodology," California Energy Commission, PIER Energy Systems Integration Program CEC‐500‐2010‐021, 2010.

[5] G. Heydt, K. Hedman, and S. Oren, "The Development and Application of a Distribution Class LMP Index," PSERC, Tempe, AZ, PSERC Publication 13–38, Jul. 2013.

[6] D. P. Chassin, J. C. Fuller, and N. Djilali, "GridLAB-D: An Agent-Based Simulation Framework for Smart Grids," J. Appl. Math., vol. 2014, pp. 1–12, 2014.

[7] P. Palensky, E. Widl, M. Stifter, and A. Elsheikh, "Modeling Intelligent Energy Systems: Co-Simulation Platform for Validating Flexible-Demand EV Charging Management," IEEE Trans. Smart Grid, vol. 4, no. 4, pp. 1939–1947, Dec. 2013.

[8] K. Anderson, J. Du, A. Narayan, and A. El Gamal, "GridSpice: A distributed simulation platform for the smart grid," in 2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013, pp. 1–5.

[9] B. Palmintier et al., "IGMS: An Integrated ISO-to-Appliance Scale Grid Modeling System," IEEE Trans. Smart Grid, no. Special Issue on High Performance Computing (HPC) Applications for a More Resilient and Efficient Power Grid, 2016.

[10] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, "FNCS: A Framework for Power System and Communication Networks Co-simulation," in Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative, San Diego, CA, USA, 2014, p. 36:1–36:8.

[11] U.S. Department of Energy, "Grid Modernization Lab Consortium." [Online]. Available: https://gridmod.labworks.org/. [Accessed: 09-Feb-2017].

[12] T. Blockwitz et al., "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models," 2012, pp. 173–184.

[13] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules. IEEE Standard 1516, 2010.

[14] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Federate Interface Specification. IEEE Standard 1516.1, 2010.

[15] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Object Model Template (OMT) Specification. IEEE Standard 1516.2, 2010.

[16] R. Fujimoto, T. McLean, K. Perumalla, and I. Tacic, "Design of High Performance RTI Software," in Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications, Washington, DC, USA, 2000, p. 89-.

[17] H. Liang, Y. Yao, X. Mu, and L. Wang, "Design and Implementation of MPI-Based Communication Mechanism in HPC-RTI," in Recent Advances in Computer Science and Information Engineering: Volume 3, Z. Qian, L. Cao, W. Su, T. Wang, and H. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, p. 121—126.

[18] A. V. Brito and A. V. Negreiros, "Allowing Large-Scale Systems Evaluation with Ptolemy through Distributed Simulation," in 2013 III Brazilian Symposium on Computing Systems Engineering, 2013, pp. 53–58.

[19] E. Noulard, J.-Y. Rousselot, and P. Siron, "CERTI, an open source RTI, why and how," in Spring Simulation Interoperability Workshop, 2009, pp. 23–27.

[20] "CERTI - Summary," 2017. [Online]. Available: https://savannah.nongnu.org/projects/certi..

[21] B. Palmintier, E. Hale, B.-M. Hodge, K. Baker, and T. Hansen, "Experiences integrating transmission and distribution simulations for DERs with the Integrated Grid Modeling System (IGMS)," in Proceedings of the 19th Power Systems Computation Conference (PSCC'16), Genoa, Italy, 2016.

[22] R. M. Fujimoto, "Time management in the high level architecture," Simulation, vol. 71, no. 6, pp. 388–400, 1998.

[23] M. Sustrik, "ZeroMQ: The Design of Messaging Middleware," ZeroMQ: The Design of Messaging Middleware. [Online]. Available: http://www.drdobbs.com/architecture-and-design/zeromq-the-design-of-messaging-middlewar/240165684#.

[24] J. Hansen, T. Edgar, and J. Daily, "Evaluating Transactive Controls of Integrated Transmission and Distribution Systems Using the Framework for Network Co-Simulation," presented at the American Control Conference, Seattle, WA, 2017.

[25] E. Ela and M. O'Malley, "Studying the Variability and Uncertainty Impacts of Variable Generation at Multiple Timescales," IEEE Trans. Power Syst., vol. 27, no. 3, pp. 1324–1333, Aug. 2012.

[26] "Arion." [Online]. Available: https://github.com/pnnl/arion.