

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.DOI

# HELICS: A Co-Simulation Framework for Scalable Multi-Domain Modeling and Analysis

TREVOR HARDY<sup>1</sup>, (Member, IEEE), BRYAN PALMINTIER<sup>2</sup>, (Senior Member, IEEE), PHILIP TOP<sup>3</sup>, (Member, IEEE), DHEEPAK KRISHNAMURTHY<sup>2</sup>, (Member, IEEE), and JASON FULLER<sup>1</sup>, (Senior Member, IEEE),

<sup>1</sup>Pacific Northwest National Laboratory, Richland, WA 99354 USA (e-mail: first.last@pnnl.gov)

<sup>2</sup>National Renewable Energy Laboratory, Golden, CO USA 80401 (e-mail: first.last@nrel.gov)

<sup>3</sup>Lawrence Livermore National Laboratory, Livermore, CA USA 94550 (e-mail: top1@llnl.gov)

Corresponding author: Trevor Hardy (e-mail: trevor.hardy@pnnl.gov).

This work was authored by Pacific Northwest National Laboratory, under contract No. DE-AC05-76RL01830; and Lawrence Livermore National Laboratory, under contract No. DE-AC52-07NA27344, for the U.S. Department of Energy (DOE); and the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, under Contract No. DE-AC36-08GO28308. Funding provided by the DOE Office of Energy Efficiency and Renewable Energy Solar Energy Technologies Office and DOE Office of Electricity through the Grid Modernization Laboratory Consortium. A portion of this research was performed using computational resources sponsored by the Department of Energy's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes. Release/publication numbers PNNL-SA-191859, LLNL-JRNL-854850, NREL/JA-6A40-87610

**ABSTRACT** As both the generation resources and load types have changed and grown over the past few decades, there is a growing need for analysis that spans traditional simulation boundaries; for example, evaluating the impact of distribution-level assets (e.g. rooftop solar, EV chargers) on bulk-power system operation. Co-simulation is a technique that allows simulators to trade information during run-time, effectively creating larger and more complex models. HELICS is a co-simulation platform that has been developed to enable these kinds of power system analysis, incorporating tools from a variety of domains including the electrical power grid, natural gas, transportation, and communications. This paper summarizes the technical design of HELICS, describes how tools can be integrated into the platform, and reviews a number of analyses that have been performed using HELICS. A short video summary of this paper can be found at [https://youtu.be/BIUiR\\_K87Wc](https://youtu.be/BIUiR_K87Wc).

**INDEX TERMS** power system analysis computing, power system simulation, HELICS, co-simulation, natural gas, transportation, multi-energy analysis, multi-domain analysis, energy system analysis

## I. INTRODUCTION

TRADITIONALLY, the electrical energy system has been conceptually divided into the bulk power system (composed of generation and transmission assets) and the distribution system, with each of those domains having specific areas of analytical interest. Analysis of the bulk power system has focused on ensuring that there is sufficient generation capacity at all times to meet the load of the system, that the capacity of the transmission network is sufficient to transport the electrical energy to the substations, and that generation resources are dispatched in an economical manner while keeping the system stable and secure in the face of typ-

ical disruptions. Similarly, analysis of the distribution system has often been most concerned with ensuring sufficient capacity in the substation transformers, distribution transformers, and distribution lines; managing voltage at the point of connection with customers; and protecting the system from the impacts of faults. The reality, though, is that these two domains are physically one system—and consequently, there are certain types of analysis that require a more integrated perspective. For example, the idea of transactive energy [1], where customers' assets respond to value signals (often based on wholesale energy prices) has been studied for many years and has recently experienced regulatory change in the

US with FERC Order 2222 [2]; transactive energy requires analysis techniques that span both bulk power systems and distribution domains.

Further, analysis needs can easily extend beyond the bulk power and distribution systems to other systems. Over the past two decades, there has been a dramatic shift in the United States away from coal as a prime-mover fuel to natural gas [3]. While coal is often delivered by train car and can be stored on-site at the generator, natural gas is transported by pipe and is much more like a just-in-time fuel source. Disruptions in the natural gas supply are not only felt more quickly (as compared to coal) but have the potential for greater impact on the electrical power system because both the natural gas and electrical networks are linked through these large generators [4].

Communications systems pose another analysis need; all areas of the electrical power system have been affected by the widespread adoption of communications technologies. Phasor measurement units (PMUs) have been widely deployed across the United States and have found use in bulk power system operation for monitoring voltage stability [5] and system oscillation detection [6]. The data required for these applications relies on effective transport over communication systems, and to understand the effects of communication system imperfections (*e.g.*, delays, out-of-order data arrival, missing data), the communication system must be explicitly modeled. Without a data-dynamic multi-domain analysis technique, it would not be possible to assess the performance of the applications and algorithms that use PMU data from such real-world systems.

To address these kinds of analysis challenges, there are two options: build a new simulator that models all the necessary domains in an integrated code base, or find a method for tying existing simulators together such that the inputs and outputs from each are coupled during runtime. From a practical standpoint, the former is generally untenable as it requires a new, integrated simulation tool for every unique combination of domains. The latter is not only much more practical but allows the use of existing tools and the (often) years of development, improvement, and validation they bring to the table. This technique is called “co-simulation” and has been in use in various forms for several decades. Past and current platforms include HLA [7], FMI [8], Mosaik [9], IGMS [10], FNCS [11] and HELICS [12]. Co-simulation has been used to address a wide variety of these emerging analysis needs: communication system impacts in managing microgrid assets for power balancing [13], transactive energy mechanisms for appropriate integration of the wholesale and retail markets [14] (even at very-large scales [15]), transient analysis [16], and DER integration and management [17]–[20].

This work focuses on the Hierarchical Engine for Large-scale Infrastructure Co-Simulation (“HELICS”), a co-simulation platform that was originally planned and defined through the work in [12], which described the design of the core co-simulation platform prior to its’ full implementation. With several years of active development, these original plans

have been implemented and expanded upon to produce a general purpose, flexible co-simulation platform. HELICS has been used in a wide variety of analyses and use cases, and is available as open-source software [21]. This work will summarize the design of HELICS as it is implemented today, including recent improvements, user-support tools that have been added to increase ease of use, and the analyses and use cases that have been conducted using HELICS.

## II. DESIGN OF HELICS

The design requirements for HELICS are described in detail in [12]; in summary, HELICS needed to be scalable, open-source, modular, cross-platform, minimally invasive, easy to integrate and use, support a broad range of simulators, and accommodate mathematical considerations such as iteration. These design requirements led directly to the software design strategy of using a layered approach in the software and a concept of a hierarchy in co-simulation topology.

### A. DESIGN PRIORITIES AND PHILOSOPHY

The following design priorities directed much of the design of HELICS and supporting tools.

- 1) Make it as easy as possible for participating simulation tools of all kinds to work together
- 2) Participating simulation tools cannot impose restrictions or requirements on other federates
- 3) Participating simulation tools should maintain control and autonomy
- 4) Implement in layered and modular architecture so as to be adaptable to a wide variety of scenarios and needs
- 5) Centralized control and/or management should be minimized

The first design priority, ease of integration, influenced the development of language bindings for many common programming languages, such as Python, MATLAB, Java, and others. It also led to continuous improvements in documentation and ease of use in the APIs and allowed conversion between data types and units in the interfaces.

The second priority (excluding tools from placing requirements on each other) requires that the timing and interfaces of one participating simulation tools not impose additional timing or interface requirements on others. This allows a great deal of flexibility in how participating simulation tools are defined at the expense of a more complicated timing coordination inside of HELICS; this flows back to the first priority. Fundamentally, HELICS allows each participating simulation tool to make the choices that work the best for it individually with the expectation that HELICS will manage the timing and data exchange complications that may arise.

The third priority, maintaining local control and autonomy, motivated HELICS to be implemented as a library rather than a runtime; this significantly affects how HELICS is used in many cases. Many simulation tools that interact with HELICS can be run as standalone executables, and having HELICS as a library means that any tool that provides

HELICS support can choose when and how it implements the HELICS APIs and how it will operate when part of a HELICS federation. This autonomy is in contrast to other co-simulation environments or platforms that than only allow operations when a specific run-time application is running. Furthermore, this autonomy requirement drove the HELICS library and APIs to provide fine-grained control over the co-simulation operations within a participating simulation tool.

The use of layers in the HELICS design philosophy (the fourth design priority) is applicable in the general software design philosophy as described above and also supports the use of a hierarchy of brokers and layers within the co-simulation itself, providing significant scalability advantages for HELICS. This is closely tied with the decision to minimize central control as articulated in the fifth design priority. (Researchers in [22] have performed a scalability comparison of HELICS with other co-simulation platforms and the results bear out the advantages of distributed time-keeping for co-simulations with many federates.)

In any coordinated co-simulation, there must be some central entity doing some coordination, and HELICS makes the conscious choice to minimize the operations performed by that entity. In each co-simulation there is an entity called the “root broker” and the root broker has two main responsibilities: 1) trigger the start of co-simulation data exchange 2) act as a last-chance router for messages and data. Note that managing the timekeeping for all participating simulation tools is not centralized in any way; this is intentionally not the role of any broker. Removing this central control was anticipated to be necessary for large scale co-simulations as it removes a major bottleneck and make parallelism across the participating simulation tools quite natural.

The timekeeping operation is intentionally distributed to allow scalability and a high degree of user control. Each participating simulation tool has a local time coordinator responsible for determining when to allow it to execute its local model and other simulation tasks. The algorithmic principle it uses is that each simulation tool can execute a particular simulation time when there is no possibility of data coming from other participating simulation tools to be produced at any simulation time prior to the time about to be simulated. That is, each participating simulation tool has causality respected and enforced by HELICS. Simulation tools requesting the same simulation time are assumed to execute in parallel, and exchanged data generated at a particular simulation time would be available at the next iteration of that simulation time or when a greater simulation time is executed. (There is a flag that can be set to allow one of these simulators running in parallel to wait until all other simulators have finished.)

Generally, a simulation time earlier than the one requested by a given simulation tool is granted when any of its inputs from other simulation tools change; when this happens, all data from that simulation time is available. However, this conservative and distributed time management strategy can, in a few cases, be very non-optimal. Therefore, HELICS

allows the use of a centralized coordinator (at user discretion) when setting up the co-simulation. Additionally, HELICS can execute asynchronous timekeeping, essentially turning off time coordination in favor of user control or when real-time management is used by all participating simulation tools.

## B. HELICS LAYERS

To meet the design requirements for a modular cross-platform design, the software for HELICS is partitioned into a series of layers with programming APIs between each. This allows development, testing, and modification of the individual layers without major concern of impact to the higher-level layers. A brief description of each of the layers is included in the following section, and a diagram is shown in Figure 1.

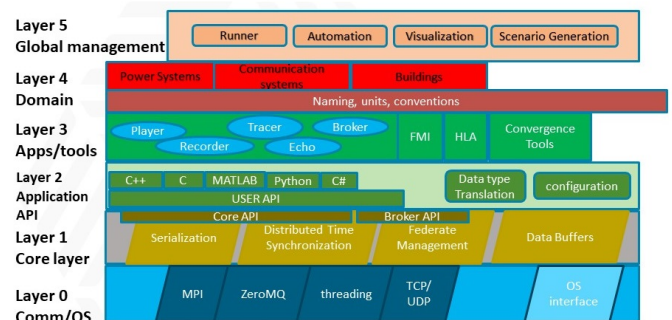


FIGURE 1: Layers of HELICS

### 1) Operating System/Communications Layer

The operating system/communications layer is responsible for the individual networking components and specific operating systems aspects. HELICS includes several messaging types between participating simulation tools, including ZeroMQ, UDP, TCP, MPI, interprocess communication, and inprocess channels; others can be added as needs arise. Support for user-defined communication channels is also planned but currently unimplemented. These communication modules enable communication between different processes, computers, and operating systems. They can be swapped with a simple change in user-exposed configuration files or command line arguments and also can be mixed and matched for specific communication channels between federation participants via the HELICS multi-broker (see section III-C). Helper objects exist to allow mixed mode communication and bridging between different networking technologies.

### 2) Core Library Layer

The core layer represents the minimum set of features necessary for a co-simulation, including time synchronization, execution control, and data exchange between simulation tools. The core layer uses the APIs of the platform layer to manage data exchange and is agnostic to the actual communication protocol in use. The core layer manages the threads used by

HELICS and handles the HELICS messages that go between the different components. The actual message transmission is left to the communications layer. The core layer handles the data management and buffering as the data flows between the different interfaces.

### 3) Application Layer

The application layer is the primary interface among application simulation tools interacting with the co-simulation framework. 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 core layer represents a generic low-level co-simulation. The application layer adds meaning to it with the inclusion of support for specific data types as well as user-defined units and types associated with the value- and message-based interfaces.

### 4) Apps Layer

The apps layer builds additional functionality on top of the application API and defines additional helper tools such as players, recorders, probes, and a broker application. In addition, there are support layers intended to allow interoperability between different co-simulation platforms such as FMI. The intention is to allow generic tools that can be used in a wide variety of situations with tighter HELICS integration than regular user applications.

### 5) Domain Layer

When working with specific domains, it may be necessary to define certain conventions in use. These conventions and any APIs around them would live in the domain layer. These could include things such as the naming conventions, connection methodologies, unit conventions, and other types of standards that may only be applicable to a small subset of users (but could be highly useful to those users) and allow tools to be swapped in and out with ease. A more detailed description is given in Section III-E.

### 6) Management Layer

The upper layer of HELICS operates on the whole of the co-simulation or set of co-simulations and is designed to help manage large co-simulations in a sensible fashion. This includes common tasks such as debugging, data management, and workflow tooling. Significant future effort is expected in this area as co-simulations become more complex and detailed with a greater number of simulation tool instances.

## C. FEDERATE TYPES

A “federate” is defined as a specific instance of a simulation tool. Multiple instances of a simulation tool (with different input data or models) may be used in a given co-simulation, and each are referenced as a unique federate. The application layer defines a few specific federate types to more cleanly separate the APIs and intent. For logical separation, the

application API differentiates “value federates,” “message federates,” and “combination federates,” which, as the name implies, uses both message and value federate constructs.

#### 1) Value Federates

Value federates are intended to replicate connections between federates on a physical level. Examples of the types of data that would typically be exchanged as values could be voltages, forces, positions, irradiance, etc. Because HELICS value connections are attempting to represent a physical reality, the connections are continuous (persistent), unidirectional, have state, have a specific data type, and optionally can be assigned a unit of measurement. (See Section II-D1 for additional discussion on value interface characteristics.) Value federates represent these value connections through specific HELICS interfaces called “publications” and “inputs.” 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 is available to directly support co-simulation FMUs. Value federates also support iterative loops (or “superdense time steps” in the FMI nomenclature) to allow federates to converge their models and reach consistency in any value connections.

#### 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 a telecommunications or computer networking exchange. Common examples are sensor measurements and control signals. Message federates capture and define interaction through data packets called “messages” which send data through HELICS “endpoints.” These endpoints send and receive data in discrete messages that are generated by a federate, pass through the federation, and then are received and used by a specific HELICS federate. No history of the packets is maintained by HELICS, and it is not persistent the way that a value interface is. HELICS acts as a courier for messages between endpoints and not an auditor or logger of those messages. See Section II-D2 for additional discussion on message interface characteristics.

#### 3) Combination Federates

While interactions can be defined in terms of HELICS value or message types, some federates may need to use both. Combination federates merge the two types and allow interactions to be defined for via either type.

#### 4) Callback Federates

In some cases, the operations of a federate is very well defined and compact, and many individual copies of that federate may be needed for a particular analysis. Examples of this could be thermostats or EV charge controllers where the control logic is simple and the number of user-defined parameters are limited. In these kinds of cases, HELICS provides a means of implementing the federate entirely using

callbacks. The interactions of a federate are defined in terms of callback operations and are executed by the core layer in a continuous fashion, simplifying the interactions and allowing large numbers of federates to be handled in a compact and efficient manner. (Note that all callback operations run in a single thread.) Callback federates may be implemented as value, message, or combination federates.

#### D. INTERFACE TYPES

The actual data transfer in HELICS occurs through “interfaces”; interfaces are external communication ports on a federate, allowing it to send messages and values to other federates. Some represent the value-based interactions and some represent the message-based interactions, as well as some that enable the crossover interactions of the two types. The core layer of HELICS defines the basic operations of the interfaces, and the application layer gives further meaning to the data transfer and tools to enable more structured and simple interaction with them. Table 1 provides a comparison of the key differences between value and message characteristics.

TABLE 1: HELICS Value and Message Characteristics

	Value Interface	Message Interface
Metaphor	Physical connection	Packet communication
Data Types	Double, complex, Boolean, vector, string, int, raw bytes	String, raw bytes
Units	Use and conversion supported	Not supported
Connections	Targeted	Targeted or addressable
Directionality	Unidirectional	Bidirectional
Connectivity	1-N or N-1	N-N
Timing	Always immediate	Delayable
Data Persistence	Persists on last sent value	Deliver-and-forget
Filters	Not supported	Reroutable, modifiable

##### 1) Publications and Inputs

Value federates interact through a publish-and-subscribe mechanism: “publications” emit values and send them to “inputs.” Inputs subscribe to publications, or publications target inputs—the connection can be defined from either the sender or receiver. The specific content of the values is arbitrary and includes explicit support for both generic data blocks (both as raw bytes as well as JSON strings) and many common types, such as floating-point numbers, integers, strings, complex numbers, and arrays. HELICS provides strict type checking between a publication and subscription and will perform unit conversion where applicable. The federate API also provides functions to query if the value on an interface has been updated since the federate last read it value, obtain the value, and note the time of the update. Classes are also available that encapsulate the interactions of a single input or publication.

##### 2) Endpoints

Unlike value publication (which generally has no specific destinations when defined), a message has a specific source,

destination, and delivery time. These messages could represent communication packets, events, or anything else that two federates mutually understand. A message federate defines “endpoints” that are the sources and destinations for the message-based federate interaction. An endpoint may also subscribe to a value-based publication, and HELICS will generate and send a message every time the value is updated. All messages must have a defined destination when sent, although as a convenience for the user, a default destination may be defined for a given endpoint. This is called “targeting,” and endpoints may be defined as “targeted” or “untargeted.” Targeted endpoints specify a target or targets on a per-endpoint basis. Untargeted endpoints may also specify a message destination on a per-message basis.

##### 3) Filters

Filters arose from a requirement to support communication simulations at various levels of fidelity without requiring that message federates alter their configuration based on the need (or not) to use a filter. This concept requires the ability of a filtering federate to insert itself into the messaging path, transparently performing some kind of operation on received messages before sending them on to their original destination (or not). HELICS defines the concept of a “message filters” (or often just “filters”) to support this functionality.

Each message filter is associated with specific 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 federate’s endpoint directly to the generator federate’s endpoint. A more complex co-simulation may require that the full communication path between the generator and controller be modeled. With the message filter functionality, filters can be inserted to convert the original message to a specific communication packet format (e.g., TCP/IP), 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 HELICS message object structure itself is such that it keeps a record of the original source and destination endpoints 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.

Filter operations are automatic, and no user interaction is required once the filter has been associated with endpoints on federates. HELICS includes a few low-overhead filter federates that provide functionality such as fixed delays, random delays, random message drop, message rerouting, and message cloning. The APIs also allow for the creation of a user-defined filter as a stand-alone federate (such as a communication system model) or via callbacks.

#### 4) Translators

While the distinct operations and definitions related to value and message federates are valuable from a conceptual point of view, in practice there are times when message federates need to interact with value federates, and defining new interfaces may not be possible. For example:

- data acquisition device that converts measurements (values) to a digital packet stream (messages)
- a control relay moves a physical control point (value) based on information received over a network (message)

This conversion of HELICS interface types is simplified through a HELICS “translator.” A translator has the role of converting packet data to value data and vice versa. A translator can be thought of as a combined endpoint, input, and publication, and can be connected as such to all other HELICS interfaces. Data sent to a translator through an endpoint results in a publication on a value interface and any publications to the input of a translator get sent out as a message to one or more predefined destination endpoints. Like a filter, all operations of a translator are automatic and transparent to the other federates. Support is included for binary and JSON based translators as well as custom user-defined translators.

#### 5) Queries and commands

In addition to the synchronized data interfaces, HELICS includes an asynchronous query mechanism (called “queries”), allowing any HELICS component to ask questions of another component. For example, a HELICS component may query available publications or endpoints, or query the entire federations structure from the root broker. Queries are useful for programmatically determining the state or configuration of the federation (monitoring) as well as allowing federates to reconfigure themselves in response to the state of the federation.

HELICS also provides an asynchronous interface allowing federates and brokers to send instructions to other federates and/or brokers; this is called the “command” interface. Built-in commands supported directly by HELICS (and thus all brokers and federates) include remote logging and debugging interfaces. Aside from these built-in commands, arbitrary commands can be defined by those creating the federates and federation to allow for customized command-and-control across their federation.

### E. PROGRAMMING LANGUAGE BINDINGS

HELICS provides language bindings in a variety of popular languages and supports federations with heterogeneous federates in this regard. The primary library is developed in C++, and a C++ API is available making use of C++17 standards [21]. A C shared library can be built alongside the c++ library to support applications requiring a simpler, C-style interface and for alternate compiler support. APIs in a number of programming languages are supported, namely Python [23], Java [24], MATLAB and Octave [25], Julia

[26], and C#. Other language interfaces are straightforward to develop because of the universality of the programming languages’ support for the C interface. Due to the language’s popularity, the Python API has a wrapper-based interface which is very similar to the C API, and a more class-oriented interface that provides the same API functionality as the C API but operating in a more Pythonic way. A similar class-oriented API is in development for Matlab.

### F. LINKAGE TO OTHER CO-SIMULATION FRAMEWORKS

Although the underlying core does not interact directly with existing co-simulation standards such as FMI and HLA, the application layer exposes interfaces for these standards. Since the designs and experience with these 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 of HELICS. Many functions defined as a part of the “FMUs for co-simulations” standard will map to the concepts in the value federate through the FMU-specific application, HELICS-FMI [27]. The “helics-fmi” application allows one or more FMUs to be loaded and run as part of a bigger co-simulation in an easy-to-use fashion that works like other federates. Future enhancements are planned that will allow HELICS federates to be wrapped as FMUs.

A HELICS federate can interact with an HLA based co-simulation through a bridge federate [28]. HLA and HELICS have different notions of data management that needs to be bridged, and though each implementation of HLA is unique, this pattern is expected to hold for most implementations.

## III. IMPLEMENTING A HELICS FEDERATION

### A. INTEGRATING A SIMULATION TOOL WITH HELICS

For a given simulation tool to be able to participate in a HELICS co-simulation, specific API calls in the HELICS core library must be incorporated in some manner into the operation of the simulation tool. There are generally two techniques for performing this integration: direct integration or wrapping the simulation tool. The specifics of the typical HELICS API calls necessary to effectively integrate the tool are discussed in Section III-B as it is their appropriate integration that allows simulation tools to be used as a HELICS federate in a co-simulation.

#### 1) Direct Simulation Integration

Direct integration of HELICS by incorporating HELICS API calls into a simulation tool’s codebase is only possible if the source code of the simulation tool is available for editing. In this case, it is possible to evaluate the operational architecture of the simulation tool and identify the points in operation where the appropriate HELICS API calls can be inserted. For simulation tools with an existing model of time (e.g., simulation tools that progress through time and update their model state at each time step), identifying the appropriate points to add the HELICS API calls is generally possible

and typically straightforward. This is also true whether the tools march through simulated time with regular timesteps or are more event-driven and simulate specific times based on events that trigger an update to the system being simulated.

As an example, the part of the hypothetical source code for a fictitious C++-based simulation tool called “GridSimulator” is shown in Listing 1. In this code you can see the headers for the “helics\_msg” library, giving visibility to important HELICS API calls such as `CombinationFederate()`, `requestTime()`, and `HelicsSubscriptionEndpoint.getMessage()`, among others. Alongside these are internal API calls that GridSimulator has defined for itself such as `model.setLoad()` and `model.runPowerFlow()`.

```
#include helics_msg.h

...
federate = new helicscpp::CombinationFederate(*config);
...
granted_time = federate->requestTime(t)
msg = (*sub)->HelicsSubscriptionEndpoint.getMessage(ep);
...
model.setLoad(bus1, msg)
pf_results = model.runPowerFlow(granted_time)
(*pub)->HelicsPublication.publish(pf_results.volt1);
...
```

Listing 1: Direct integration of the HELICS APIs into the fictitious C++-based simulation tool GridSimulator.

## 2) Wrapper Integration

Wrapper integrations are typically required in two specific cases: 1) the source code for the underlying tool is not available, and control of the tool is only provided via an API or 2) the underlying tool is more of a library of relevant functionality rather than a full-fledged simulation tool and needs supporting code to create actual simulation functions (*e.g.*, time advancement). In either case, the role of the wrapper code is to act as a bridge between the core simulation tool and the rest of the HELICS federation. This bridge wrapper is realized in a language that has support for both the simulation tool’s APIs and the HELICS APIs and is responsible for controlling the simulation tool while facilitating the synchronization and data exchange with the rest of the federation.

Listing 2 shows an example of the use of a Python script to provide this wrapper integration under the assumption that the fictitious simulation tool GridSimulator has an appropriate API with a Python library/module. Though there are strong similarities to the APIs used in the direct integration example shown in Listing 1; this sample code uses the public APIs made available by both GridSimulator and HELICS. The API provided by GridSimulator must be sufficiently featured to allow the wrapper to do things like update model state based on data received from the co-simulation federation, control the flow of simulated time in the model, and extract information from the model to publish it to the federation.

```
import helics as h
import gridsimulator as gs
```

```
...
while time < maxtime:
    granted_time = h.helicsFederateRequestTime(time)
    control_signal= h.helicsEndpointGetMessage(endpoint)
    gs.set_load(model_obj, "bus1", control_signal)
    pf_results = gs.runPowerFlow(model_obj, granted_time)
    h.helicsPublicationPublishDouble(pf_results.volt1)
...
```

Listing 2: Python-based wrapper integration of the HELICS APIs into the fictitious simulation tool GridSimulator.

As previously mentioned, if the underlying simulation tool is more of a library than a fully formed tool, the wrapper must also take on additional responsibility to create the necessary functionality expected of this type of simulation tool. MATPOWER [29] is a good example of this; MATPOWER provides a library with API calls to formulate and solve a variety of power system problems but itself has no sense of time. To create a MATPOWER-based simulation tool, the developer must add functionality to create and update model state as a function of simulated time. (This has been accomplished as a part of the HELICS project through the creation of a MATPOWER wrapper [30].) When comparing code in Listing 1 vs Listing 2, the former makes no mention of time, implying that GridSimulator is managing this itself while the latter includes an explicit `while` statement that advances time and model state.

## B. HELICS FEDERATE LIFECYCLE

Integration of a simulation tool with HELICS is necessary to allow it to be used as a HELICS federate and is realized through the process of calling the appropriate HELICS APIs to move the federate through its lifecycle. The following are the sequential progression of states that a federate is required to step through, as shown in Figure 2 with each stage marked by the use of one or more HELICS APIs. All federates in a HELICS-based co-simulation go through this same lifecycle, although the specific APIs used depend on the functionality the federate provides. Sections III-B1, III-B2, III-B3, and III-B4 discuss the details of each of these stages.

### 1) Creation

Federate creation is the process of registering an executable as part of a HELICS federation (co-simulation) and configuring the simulation time synchronization process and data exchange interfaces; this defines the HELICS “creation state.” The HELICS library provides APIs to allow these configuration steps to be done programmatically. For simulation tools where the user does not or should not have access to the source code, it is also possible to access the same configuration details via an externally defined JSON file. Both techniques are demonstrated in Listing 3. In cases where information from other federates is required for defining the interfaces, the entry into “initializing mode” can be done iteratively, allowing a state where all federates are registered and requested initializing mode but some may be returned to the “creation” state to add additional interfaces or connections.

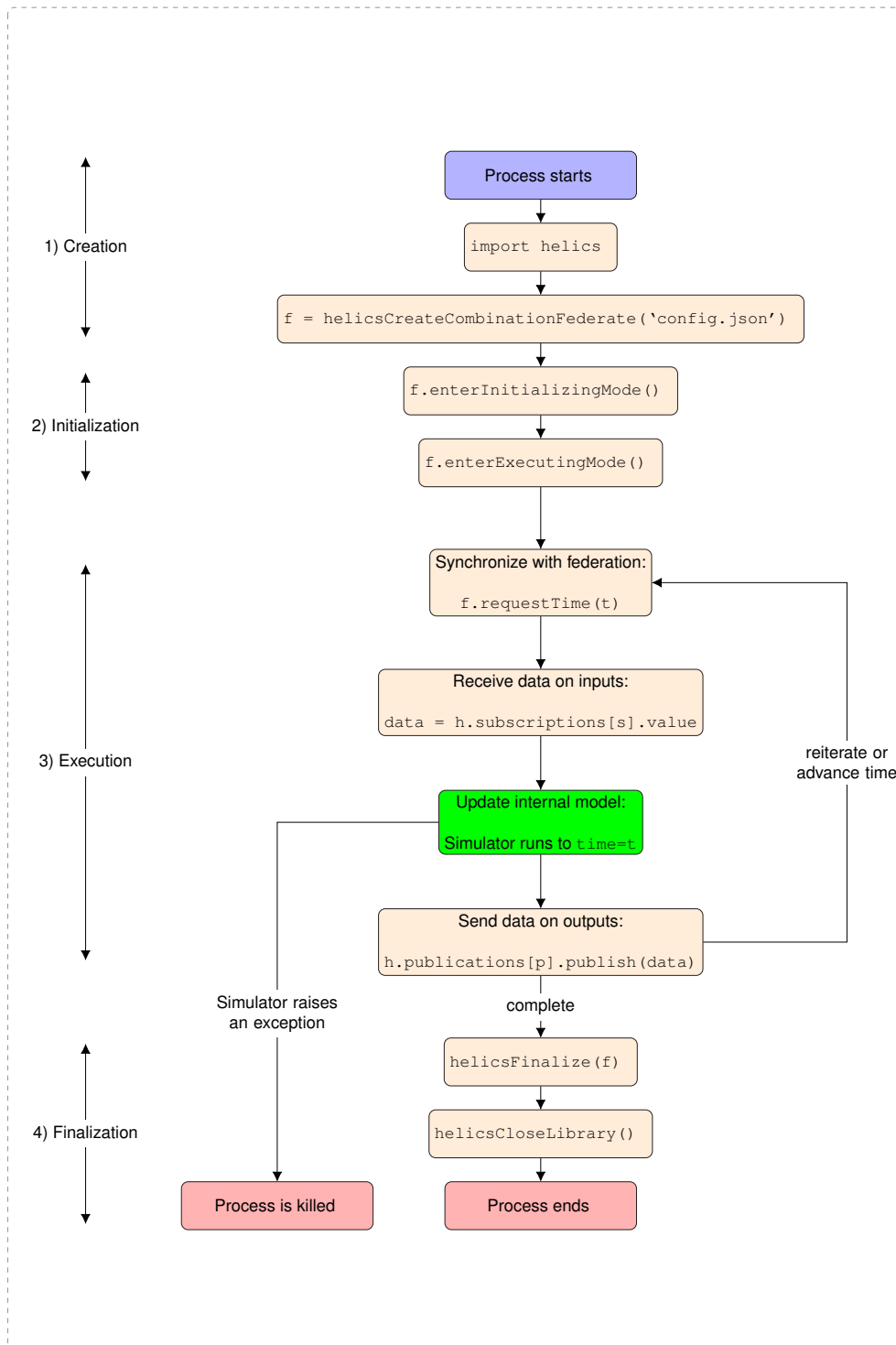


FIGURE 2: Lifecycle of a HELICS federate

```
fed = h.helicsCreateCombinationFederateFromConfig("config.json")
# OR
fi = h.helicsCreateFederateInfo()
h.helicsFederateInfoSetFlagOption(fi, h.
    HELICS_FLAG_ONLY_TRANSMIT_ON_CHANGE, True)
fed = h.helicsCreateValueFederate("name", fi)
sub1 = h.helicsFederateRegisterSubscription(fed1, "pub1")
pub1 = h.helicsFederateRegisterPublication(fed1, "pub1", h.
    HELICS_DATA_TYPE_STRING, "")
```

Listing 3: Two alternative methods of configuring a HELICS federate during the creation stage of the federate lifecycle.

## 2) Initialization

Once registered with the federation and configured, the federate makes the API call `helicsFederateEnterInitializingMode()` to enter initialization mode. Initialization mode exists to help a federation prepare to begin the advancement through simulation time. This may involve iterative data exchange with other federates to reach a collective consistent state, loading in historical or state data to initialize its internal model, or simply waiting for other federates to do any of the above. Federates may also use initialization mode to publish values that will be available to all federates in initialization mode or, for those not involved in iteration, at simulation time 0 in the main simulation step (execution mode, see Section III-B3). It is not required that federates enter initialization mode, and any that choose to skip it will have no visibility to the activities taking place by federates that are using it to initialize their models.

## 3) Execution

Once the federate has been created, all subscriptions, publications, and endpoints have been registered, and the federation initial state has been appropriately set, federates enter the main co-simulation mode by making a call to `helicsEnterExecutingMode()`. The HELICS simulation time is set to zero upon entering this mode, and any values published or messages sent during initialization mode will be available to the federation upon entering execution mode. `helicsEnterExecutingMode()` can be considered a barrier to the beginning of the co-simulation proper. After making a call to `helicsEnterExecutingMode()`, the main execution of a given federate is blocked until all other federates on which it depends also make the same API call. Typically the topology of the data exchange between federates is such that all federates must make the call to `helicsEnterExecutingMode()` before any of them begin executing the co-simulation.

Though the exact structure of the HELICS API calls by a federate is a function of both the software architecture of the underlying simulator and the particular use case being run, generally there is a four-step loop each federate runs as it advances through simulated time. These steps are shown in the "Execution" portion of Figure 2.

### 1) Synchronize with federation:

`helicsFederateRequestTime()` is used by federates to request a specific time to which it will advance its internal model, thereby bringing it into synchronization

with the rest of the federation. The requested time is defined by the dynamics of the internal model and/or the typical simulation execution pattern of the underlying simulation tool. For example, a particular tool may have a strict timestep of one second and expects its model to be updated this frequently. Alternatively, the model may be more stateless by nature and only need to update when one of its inputs changes.

Though a federate may request a given time, HELICS may grant an earlier time if there is new data on any of the input interfaces the federate has defined. It is the responsibility of the federate to determine what it will do with these inputs (*e.g.*, update its internal model, ignore them), and there are configuration options to help manage these early time grants to allow for more computationally efficient federate operation.

2) **Receive data on inputs** Once granted a time, a federate will typically check value inputs and message endpoints to collect any new data that defines the current state of the rest of the federation using APIs such as `helicsIntGetDouble()`, `helicsInputGetJSON()` (value interfaces) and `helicsEndpointGetMessage()` (message interfaces). These are effectively the boundary conditions of the federate that intersect or overlap with the rest of federation and contain the latest state data sent by them.

3) **Update internal model** Given the federation state at the federate's interfaces, it is likely the federate now has an inconsistent internal state with these boundary conditions as defined by the information it just received from the rest of the federation. To resolve this, the federate will typically recalculate the state of its internal model using these new boundary conditions. This effectively brings the federates internal model up to date with the rest of the federation as of the granted simulation time. As these are processes internal to the federate and its model, there are no relevant HELICS APIs for this process; instead, tool-specific APIs and functionality are typically used.

For example, a distribution system federate may have an input that represents the substation voltage as defined by a bulk power system federate. When the bulk power system federate solves and publishes out a new voltage for the substation, the distribution system federate will see that change, update the substation voltage in its internal model, and re-solve the distribution system powerflow. This brings the distribution system's internal model (*e.g.*, line flows, nodal voltages) into a consistent state with the substation voltage.

4) **Send data on outputs** Once the internal state has been updated, the federate will send out a subset of its internal state variables to the rest of the federation using HELICS APIs such as `helicsPublicationPublishDouble()` (value interfaces) or `helicsEndpointSendMessage()` (message interfaces). The specific variables that are distributed are those

defined in by the configuration of the federate and are generally boundary conditions for other federates.

Given the necessarily circular nature of the dependencies in the data exchange of the federates, it is typical for the publication of values from one federate to trigger an update in another, which in turn publishes new values back to the first. Depending on the analysis requirements and the capabilities of the federates involved, this circular dependency can be resolved by the federates continuing to publish, get new inputs, recalculate their internal model and publish again until a sufficient degree of consistency has been reached between the necessary federates. This process is called “reiteration” or “co-iteration,” and HELICS provides APIs to help enable this convergence.

#### 4) Finalization

Once the federate has completed its contribution to the co-simulation and simulated all necessary time, it needs to close out its connection to the federation using the API `helicsFederateFree()`, signaling to the core and brokers that the federate is leaving the co-simulation. Finally, once the federate has completed finalization, `helicsCloseLibrary()` is called to cleanup and close the HELICS library.

### C. FEDERATION TOPOLOGIES

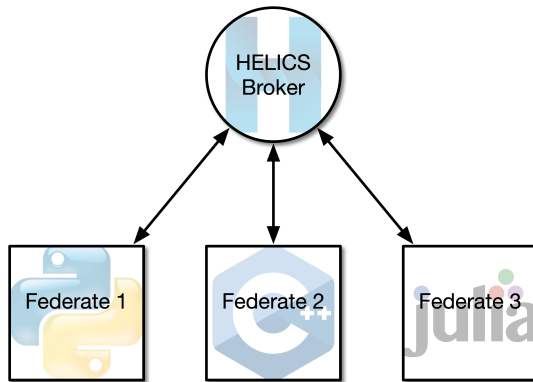


FIGURE 3: Single-broker federation

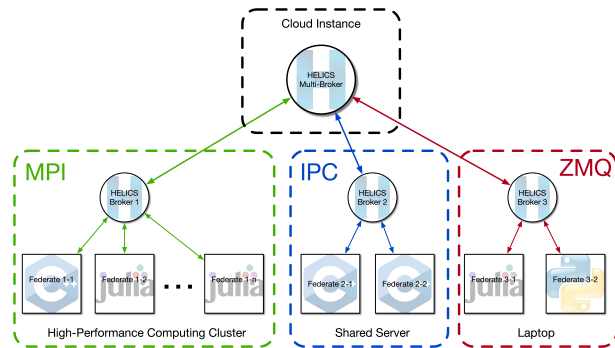


FIGURE 4: Broker hierarchy with multi-broker to span HELICS core types

HELICS supports a distributed architecture for co-simulation that allows for HELICS federates to exist and run in distinct computing environments on different computing hardware (which may be in very different geographic locations) as long as the communication protocol among the hardware is supported by the HELICS core. This allows for the creation of hierarchical co-simulation topologies linked together by HELICS brokers.

In order to run a HELICS federation, there must exist at least one HELICS broker that can communicate with all the federates, at least indirectly; this is called the “root broker”. Figure 3 shows the most commonly used “1 broker”  $\iff$  “N federates” architecture. It is also possible to set up a co-simulation that uses “1 broker”  $\iff$  “M sub-brokers”  $\iff$  “N federates” as seen in Figure 4. Doing so places a broker over a set of the federates, with federate groups typically defined by a common computing environment. This layered architecture can be extended to support an arbitrary number of layers in the hierarchy. For example, one federate group may all be on a particular institution’s high-performance computing cluster, while another is on a separate server at the same institution, while a third is on the laptop of an entirely separate institution, with the root broker running in a cloud instance. Each local broker allows for low-overhead communication between its federates while also supporting communication to the federation as a whole as needed. Ideally, federates under a single broker are most likely to need to exchange data with each other and less likely to need to exchange data with those under other brokers, because the communication overhead is higher. Careful design of the broker hierarchy can help mitigate slowdowns due to network latency.

The segregation of federates under any number of brokers also allows for explicit configuration of each sub-federation to accommodate any particular networking or communication challenges particular to a given computing environment. For example, looking again at Figure 4, the portion of the federation operating in a high-performance computing environment may want to use that cluster’s available MPI hardware and thus use the HELICS MPI core. Those that are running in a single server may all be written using the Boost library and thus are able to use memory-based sharing in the IPC core, while those on the laptop may just use the default ZMQ core. HELICS is able to bridge these sub-federations through the use of a “multi-broker” at the root broker, allowing for more optimal data exchange for each computing environment.

### D. HELICS SUPPORTED SIMULATION TOOLS

Though the immediate application of HELICS is in power system applications, HELICS is a general co-simulation platform and can support data exchange between a variety of simulation tools. Figure 5 shows many of the wide range of tools with HELICS support (currently or planned), while the following section highlights some of the most commonly used tools that are known to have HELICS support and have

been used in one or more demonstrations or studies.

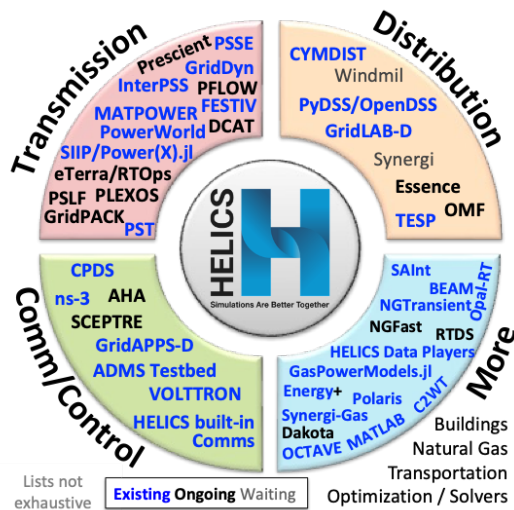


FIGURE 5: A wide range of simulation tools with known HELICS interfaces

#### • Transmission

- PowerWorld - Steady-state power flows and optimal power flows in [31]
- P/SSE - Transient studies in [17]. Steady-state and electromechanical dynamics implemented through PyPSSE [32]
- MATPOWER/MOST - Steady-state power flows, optimal power flows, multi-period optimization [30]
- Andes - Python-based power system simulation tool with many built-in models [33] used in [34], [35], [36]
- PYPOWER - Steady-state power flows and optimal power flows is a supported tool in TESP [37]
- SAIInt (Electric Network) - Steady-state power flow and optimal power flow [4]

#### • Distribution

- GridLAB-D - Steady-state [38] and transient studies [17]. Examples of use include large-scale transactive energy studies [39], microgrid transactive energy studies [38], smart grid communication studies [40] and [41], microgrid DER integration studies [42].
- CYME - Steady-state and time series implemented through CYMEpy [43]
- OpenDSS - Steady-state and time series analysis implemented through PyDSS [44]. Examples with PowerWorld in steady-state [31] and transportation in BEAM [45].

#### • Communication

- HELICS built-in filters for delays, dropped packets, etc. E.g. evaluating transmission impact of

AGC delays from DERs [34] and EV smart charging [35]

- ns-3 - Evaluates impacts on distribution system operation due to communication effects [46]
- Omnet++ - HELICS integration with supporting examples [47]

#### • Natural Gas

- SAINT - Dynamic and steady state hydraulic gas simulations coupled with grid in [4], [48]
- NGTransient - Evaluates natural gas pipeline physics [49]

#### • Other

- BEAM - Transportation simulator, used to explore impacts on distribution in [45]
- Caldera - Provides EV charging profiles for integration with power system simulators [50], [51]
- OpalRT - Real-time power system simulator used for hardware-in-the-loop in [52], [53]
- EnergyPlus - Multi-zone commercial is a supported tool in TESP [37]
- Ochre - Residential model building simulator [54]

### E. STANDARDIZED INTERFACES

To assist in the interchangeability of similar simulation tools in a HELICS-based co-simulation, a few standardized use cases and interface definitions have been developed. For example, one use case that has been defined is a steady-state transmission and distribution system powerflow. This use case makes the following data-exchange requirements between the two federates:

#### • Transmission Required Interfaces:

##### -- Inputs:

- \* Distribution system loads at all points of common coupling

##### -- Outputs:

- \* Transmission system bus voltages at all points of common coupling

#### • Distribution Required Interfaces:

##### -- Inputs:

- \* Transmission system bus voltages at all points of common coupling

##### -- Outputs:

- \* Distribution system loads at all points of common coupling

Although it may seem redundant for simply defined use cases such as these, the mirrored definitions of the interfaces allows the developer of the use case to confirm that every input required by one federate has that met by an output from another federate. Furthermore, each of the input and output interfaces calls out a specific interface definition associated with each federate type (not shown in the definition above). For example, the distribution system federates load output interface definition includes the following:

- **HELICS interface type:** value
- **HELICS interface name:** pcc.<GUID>.pq
- **Units:** MVA
- **HELICS data type:** complex vector with a single complex value for each phase (typically three phases).
- **Tags (metadata):** phases

The GUID in the interface name is intended to be related to names from the model in question and thus allows the definition to be applicable beyond any particular model file. HELICS supports metadata tags that can be associated with any interface; in this case, the “phases” tag is used to indicate which phases are represented in the interface and allows the receiving federate to comprehensively understand the data that is being sent. For example, queries on the tags associated with this interface will indicate how many elements are in the complex vector that is being sent by the distribution federate and which phases correspond to which elements. The standardized definition of the phases tag for this purpose has also been defined.

The standardization of the interfaces also makes it clear when a federate may have to do some additional work to support a given use case. For example, the distribution system load interface at the point of common coupling with the transmission system federate will produce a three-phase unbalanced load as a vector of complex values. It is common for transmission system simulation tools to only use positive-sequence values in calculating their powerflows. The standardized interface definition makes it clear that there must be a conversion from three-phase unbalanced to balanced power before applying this value internally in the simulation tool.

## F. MANAGING HELICS-BASED CO-SIMULATION

Depending on the number of federates in the co-simulation, launching it may or may not be a challenge in and of itself. Some tools can only be run from a GUI, and others only on the command line. As the number of federates increases, the chore of launching the co-simulation becomes more time consuming, and simply managing its operation becomes more complex. The HELICS teams has developed a number of tools to help with these challenges.

### 1) Co-simulation Launching with “helics\_cli”

“helics\_cli” is functionality distributed as a part of Py-HELICS (the Python language binding) that provides several useful functions. The most popular of these is the ability to launch a co-simulation, collect log messages, and write them out to log files. “helics\_cli” takes a JSON file as an input, the contents of which define the federates to launch, which command-line calls to execute when launching them, and if it should generate a broker for the co-simulation or let another process handle it. After writing this JSON file, launching the co-simulation is handled with a simple single command: `helics run --path=<path to runner JSON>`. If any federate crashes during the co-simulation, “helics\_cli” ensures all other federates close down cleanly as well. Though only applicable for federates that can be launched from the

command line, it is the preferred method of launching a co-simulation.

### 2) Co-simulation Data Collection with “helics\_cli”

In addition to the launching functionality, “helics\_cli” provides the ability to attach an observer federate that collects all the value and message outputs and writes them to a sqlite database. By default, the observer collects all output data from all federates, but this can be a useful artifact in debugging as well as a means of sharing results with others by simply sharing the database file. sqlite is not intended for large amounts of data and will not be appropriate for all co-simulations. Additionally, adding an observer to a federation may produce some degree of co-simulation slowdown if the observer must write large volumes of data to the database and must contend with network and/or local disk delays.

### 3) Co-Simulation Management

HELICS provides two methods of interacting with a running co-simulation, and both use web technologies. The first is a REST API that the HELICS broker provides; it offers a set of queries that can be made to understand the topology and state of the co-simulation. The API includes access to any queries allowed in the system, as well as commands for debugging and federation control II-D5. Brokers can also be created through the API as part of the Broker server, which can generate brokers on demand.

The second method is through “helics\_cli”. “helics\_cli” provides a web-based GUI with additional functionality. When launched, the GUI allows the user to see the composition of the federation, both in terms of federates as well as their interfaces, launch the co-simulation, pause the co-simulation, and evaluate the simulation state of a federate as well as the most recent values and messages being sent by the federation. Currently, the web GUI uses the “helics\_cli” observer capability allowing all these values to be written out to an sqlite database; this database can be re-loaded by the web GUI at a future time to inspect a co-simulation that took place in the past. At this time, the web GUI is best suited to smaller federates with more limited federate and interface counts.

## IV. APPLICATIONS OF HELICS-BASED CO-SIMULATION

### A. OVERVIEW OF APPLICATIONS

Flexibility is one of the key requirements in the design of HELICS [55] (see Section II-A), and it can support a wide range of co-simulation use cases across many different fields. This section highlights a number of past and ongoing efforts using HELICS to demonstrate this breadth and provide a starting point for readers who might be working on similar project areas.

Given HELICS’s roots as a transmission-distribution-communications-markets simulation framework for electric power systems [12], many of the example applications come from power grid use cases such as transmission-distribution interactions, advanced grid control schemes, and

even hardware-in-the-loop testing. However, a wide range of other applications that involve co-simulation beyond the power system can also readily be supported in HELICS. These include multi-infrastructure applications and other applications as highlighted below. In addition, a much wider range of applications are possible, and future researchers are encouraged to use HELICS in more diverse ways and apply it to even more fields.

## B. TRANSMISSION-DISTRIBUTION INTERACTIONS

The continued increase in deployment of distributed energy resources (DERs), electrification of end uses—especially transportation—and increased opportunities for demand-side resources to participate in wholesale markets (*e.g.* FERC-2222 [2]) have all created an unprecedented need to better understand and simulate the interactions between the transmission and distribution portions of the electric grid. This includes everything from market-timescale interactions to questions around higher-speed engineering phenomena such as frequency response or electromechanical stability.

### 1) TSO-DSO interactions

#### a: Price-responsive demand

A number of past co-simulation efforts have looked at the interactions between wholesale electric power markets and demand that responds to price signals. For example, [56] compared simulations using transmission only (with demand price elasticity) versus using a full transmission and distribution co-simulation with individually price-responsive loads. They found that the transmission-only simulation failed to capture key oscillations and differences in load profiles versus fixed demand. Moreover, the transmission-only simulation introduced some erroneous price spikes that were not present with the more detailed transmission and distribution co-simulation.

In a separate and larger study, Hansen, *et al.* [14] evaluated integrated wholesale and retail markets with price-responsive DERs for 15,000 distribution systems with over one million DERs. In particular, the study found that at high levels of real-time energy market participation by DERs, it is best for DERs to bid their demand into wholesale energy market. If instead DERs simply respond to real-time prices as they clear (so called “prices-to-devices”), oscillations in physical power system and in market clearing price develop (see Figure 6).

#### b: Transmission-Distribution-Market Platforms

Given the interest in the interactions between wholesale and retail markets, multiple testbeds that pre-couple a fixed set of simulation tools have been developed, each with a somewhat distinctive objective. Here we introduce three such efforts. Rather than competing with HELICS, these frameworks provide a higher level of abstraction. Notably, two were originally developed with semi-custom co-simulation frameworks and have since been adapted to use HELICS for enhanced coordination and more modular interfaces to component tools.

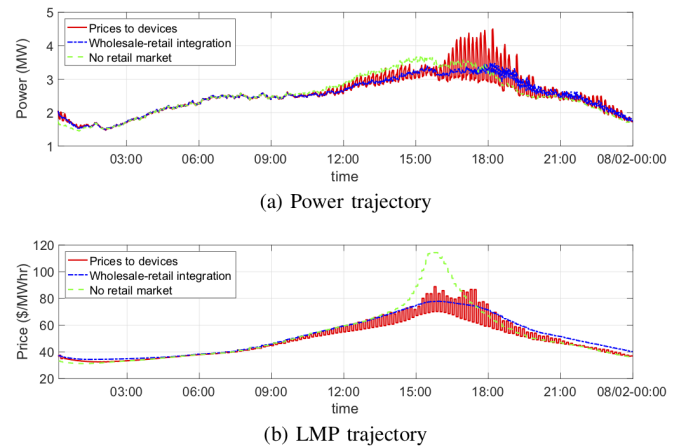


FIGURE 6: Results from [14] showing the impact of DER participation in an integrated transactive wholesale-retail market when DERs do and do not bid their demand into the market. Without bidding, oscillations develop in the power system and market clearing price.

The Integrated Grid Modeling System (IGMS) [10] represents an early example of this type of platform. It captures ISO-to-appliance scale simulation by bringing together highly detailed wholesale market operations in FESTIV [57] with transmission-scale power flow in MATPOWER [29] and dozens to thousands of distribution system simulations each running in a separate instance of GridLAB-D [58]. In this context, FESTIV provides multiple nested timescales for wholesale markets including day-ahead and intra-daily security-constrained unit commitment, real-time security-constrained economic dispatch, and automatic generator control (AGC) estimation to capture actual seconds-scale commands and (imperfect) response of generators to meet regulation and other reserve product demands. GridLAB-D provides both distribution-scale 3-phase unbalanced power flow and simplified models of buildings with end uses, including thermal models for weather dependence, a wide-range of individual appliances, and mechanisms for price-responsive control.

Originally, IGMS used a custom Python and MPI-based set of scripts to orchestrate the co-simulation [18], and the challenges with such an approach helped to inform the design of the HELICS platform. Later, the IGMS platform was ported over to use HELICS for co-simulation instead, resulting in faster performance and significantly improved modularity and scalability.

The Transactive Energy Simulation Platform (TESP) [59] was developed to provide a means of evaluating transactive energy mechanisms using a common suite of simulation tools such as GridLAB-D, PYPOWER, and Energy+. Users build their own custom transactive agents operate the system performing tasks such as DER management (*e.g.* HVAC systems or EV chargers), running retail markets, or operating the distribution system. TESP has been used for several studies

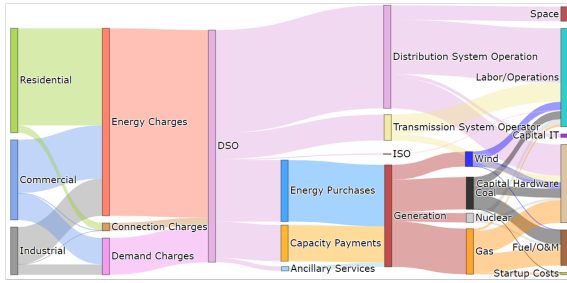


FIGURE 7: Based on the comprehensive bulk power system and distribution system modeling, simulation, and analysis in [39], the flow of money from customer to generator in today's power system can be more easily summarized.

and includes sample agents, models, and datasets from these studies as starting points for new users of the platform. Further details can be found in Section IV-D6.

Recently, TESP was used to implement an integrated wholesale-retail day-ahead and real-time energy market controlling HVACs, electric water heaters, and batteries in a study called "Distribution System Operator + Transactive" or "DSO+T". This study comprehensively evaluated the impact of widespread adoption of a transactive energy system in ERCOT, looking at both the technical and economic impacts under the ERCOT power system as it exists today and in a hypothetical high-renewable future. Full results can be found in [39]. Figure 7 shows the flow of money from consumer to generator; this perspective would only be possible through the fully integrated bulk power system and distribution system modeling with wholesale and retail market operations enabled by co-simulation.

Another recent platform [54] focused on high-fidelity simulation of the distribution-level aspects of transactive energy schemes, notably including those based on distributed ledger approaches such as blockchain. Here, detailed home models are included with a combination of operational, controllable, high-resolution residential energy (OCHRE) simulators for each home and the foresee<sup>TM</sup> home energy management system (HEMS) to coordinate a home-level, rather than individual appliance, participation in the transactive market. The HEMS also coordinates DERs including solar and storage. OpenDSS provides a simulation of the distribution power flow and utility equipment. A generic distributed-ledger smart-contract mechanism then links these to various market configurations for comparison.

A recent study with this framework compared a fairly traditional double-auction market clearing (similar to wholesale electricity markets) to peer-to-peer approaches and found that in nearly all cases, both markets provided customer energy cost savings relative to net metering tariffs. For the simulated utilities, both markets provided increased revenue with high DERs, but lower revenue with low DERs, compared to net metering. Of these, the double-blind market with high DERs was noteworthy for providing a modest increase in utility income while increasing cost savings for a large fraction

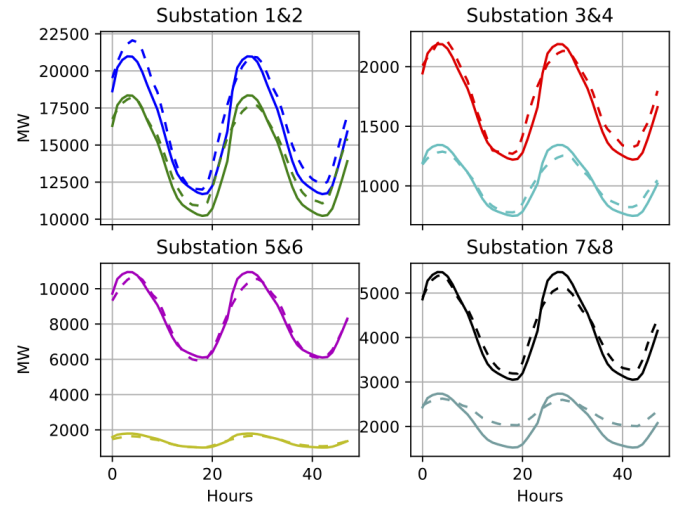


FIGURE 8: Results from [60] showing difference between using full-fidelity (dashed lines) and simplified (solid lines) distribution physics models in an integrated wholesale-retail transactive system.

of customers. Moreover, every customer on the feeder (including those without DERs) saw cost savings in all cases relative to the self-consumption-favoring tariffs being rolled out in some states. Additional results along with detailed descriptions of the co-simulation implementation including timing can be found in [54].

#### c: Improving TSO-DSO co-simulations

Some past efforts have also looked into ways to enhance TSO-DSO co-simulation performance or accuracy. For example, [60] developed a simplified representation of price-responsive DERs in an integrated wholesale-retail transactive system. These analysis typically use co-simulation to bridge the physical and market models in wholesale (transmission) and retail (distribution) systems. The simplified model reduced the data transfer over the co-simulation bus and the computational load in modeling the physics and market behavior in the distribution system.

For improving accuracy and real-world applicability, it is also important to expand the research-oriented co-simulations above—which typically duplicate one or more test feeders and connect them directly to the transmission bus—to use full-scale, heterogeneous distribution system models that capture (or mimic) the large diversity and full topology seen in the grid. Ideally such underlying data would also capture substations, sub-transmission, and other parts of the grid often left out in the examples described above. Reference [31] represents a start in this direction by using HELICS to co-simulate hundreds of synthetic separate distribution feeders, substation, and subtransmission systems in OpenDSS connected to synthetic transmission systems running in PowerWorld.

#### d: Future TSO-DSO Directions

In addition to further using such methods to explore future grid scenarios and inform related decisions, there remain a number of opportunities for future research extensions in this application area, including:

- Working out challenges associated with convergence at the transmission-distribution interface. In this area, past work has shown that for some applications, taking advantage of co-iteration to ensure consistency of boundary conditions can play an important part in simulation accuracy. However, this additional back and forth can not only can be slow, but also may never converge. Ongoing work is looking at ways to speed convergence through approaches such as gradient descent and heavy-ball heuristics to overcome convergence challenges particularly with electrically linked/looped distribution systems connected to multiple transmission nodes
- Integrating co-simulation into industry practice. A particularly promising idea in this space would use HELICS to enable operators of two parts of the grid to conduct synchronized simulations without sharing proprietary models. For example, an independent system operator (ISO) could run time-series simulations of upcoming market operations using their own tools and link with HELICS through the Internet to distribution simulations run by one or more local utilities. ISO New England is an early adopter of HELICS and has begun using HELICS in an exploratory fashion to help understand the impact of distribution load characteristics on bulk power system transient response.

#### 2) Co-simulation With Faster Dynamics

In addition to the typically quasi-steady-state timescales of TSO-DSO interactions described above, the growth of DERs in general, and in distribution-connected inverter-based resources (IBRs) requires careful coordination at electromechanical dynamics (and faster) timescales. Here again, co-simulation can be useful for scaling, automation, integrating the effects of communication latency on control-schemes, and more.

GridLAB-D's motor dynamic models and inverter models support traditional transient analysis with a number such studies. [61] Coupled PSS/E with GridLAB-D to evaluate the impacts of a bus-fault event, comparing the performance of the high-inertia induction motors (transmission-connected) and low-inertia induction motors (distribution connected). A similar study in [62] looked at the performance of IBRs when operating as grid-forming inverters, showing dramatically improved response to a transmission-connected-generator trip, particularly at very high penetrations (see figure 9).

Recent work with the Cyber-Physical Dynamic Simulation (CPDS) testbed highlights this potential with a focus on frequency response by DERs. [63] describes this framework's use of ANDES [33] as an open-source transmission-scale dynamics simulation platform coupled through HELICS to multiple instances of OpenDSS. This work also validates the

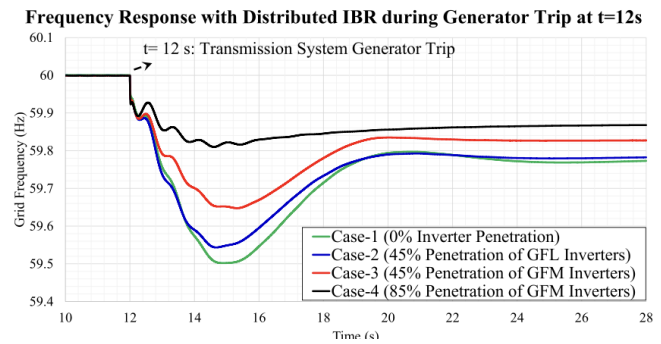


FIGURE 9: Results from [62] showing the impact of inverter-based resources on transient response in a HELICS-based co-simulation.

co-simulation vs. a combined, single-tool model showing a very close match. It further highlights how the co-simulation computation time is nearly identical for the small test systems while also scaling well (much less than linearly) to much larger-scale systems (*e.g.*, 2,000 transmission nodes and 1 million distribution nodes) through the parallelization enabled by the HELICS-based co-simulation.

In addition, co-simulation has been used to automate full-interconnect scale dynamic simulation in a real-time context within an ISO-like energy management system (EMS) framework. Specifically, in [64], HELICS was used to automate the updates to a commercial eTerra EMS system in conjunction with an online transient stability analysis tool (TSAT) developed by PowerTech to evaluate the potential to use the composite load model within WECC-wide on-line stability analysis.

#### C. RESILIENT DISTRIBUTION SYSTEM MODELING

A resilient distribution system (RDS) uses local resources, such as customer-owned solar or community-owned battery storage, to prepare for disturbances and more rapidly recover from system events—for example, by linking hospital backup generators and community solar to extend the operational time of the hospital during a disaster. This requires new control and communication systems, sometimes aggregated into microgrids, that can either provide services to or operate independently from the bulk system. This coordination, both between the bulk system and RDS or within the RDS, is being tested in simulation prior to deployment to maximize performance and integration of DERs.

CleanStart DERMS uses HELICS to evaluate the potential for achieving black start and restoration objectives through ad-hoc microgrids powered by DERs [65]. The goal is to implement a DER management system that can start a microgrid after an outage, then slowly re-energize the rest of the system by integrating additional DERs and nearby generation. Coupled models of distribution, sub-transmission, and communication systems, along with multiple dispatch, restoration, and control layers, are being tested to identify issues and improve the effectiveness of the control systems.

Communication layers are implemented to understand the robustness of the system to lost or limited data, assuming that the system will need to operate during extreme conditions.

Co-simulation was used to evaluate the performance of a resilience-based transactive system for coordinating centralized control and distributed field devices in partnership with Duke Energy. Operations during “blue” sky and “gray” sky conditions were evaluated to determine the effectiveness of the control design in conditions that are challenging to replicate in the field. The Duke-RDS project was a proof of concept that successfully demonstrated that coordination of distributed assets, using existing commercially off-the-shelf relays and open-source software, can produce a more flexible system [66].

The National Renewable Energy Cooperative (NRECA) is using HELICS to support machine learning for energy theft detection [67]. To do so, NRECA built distribution system models connected to agents that control loads in the distribution system to simulate energy theft. These simulations generate large-scale datasets that in turn are used to train models using machine learning. Using HELICS to link the power system models and the energy-theft agents allows the location and behavior of the energy theft to be easily modified, allowing the creation of large, robust datasets that provide a better training environment for machine learning. This higher-quality training produces higher-quality detection models.

#### D. ADVANCED COORDINATION AND CONTROL SCHEMES

It is expected that advanced control systems will continue to be integral to the growth of the future power system, particularly in light of high penetrations of distributed resources. These controls will often cross traditional boundaries from behind-the-meter building controls to distribution systems to transmission operators, and will include communication systems operated by a variety of owners. It is important to understand how all of these new controls will interact with each at the system scale to ensure stability, cost effectiveness, and the desired outcome. Here again, co-simulation can help.

##### 1) CyDER

In one example, the increasing complexity of transmission system interactions makes it helpful to automate the integration of data from multiple sources into Energy System Management applications. The also applies at the distribution level for management of DERs. The Cyder project [68] used FMI modules to develop a plug-and-play co-simulation for simulating operation and control in high-penetration DER scenarios and was an early test case for FMI capabilities within HELICS.

##### 2) Citadels

The Citadels project is a US Department of Energy (DOE) Grid Modernization Laboratory Consortium (GLMC) project

that was designed to increase operational flexibility between multiple microgrids using peer-to-peer control over an OpenFMB messaging bus through a consensus design algorithm on commercial hardware. To test the algorithms before deployment on the demonstration hardware, a power-communication-control co-simulation was developed to evaluate the performance of the microgrid and device control agents over a variety of scenarios (see Figure 10 for a federation architecture diagram). HELICS linked GridLAB-D, ns-3, and custom control agents developed in Python to test variations on distributed optimal power flow and collaborative autonomy algorithms to maximize DER utilization, both under normal operations and with degraded communications. By utilizing the co-simulation platform, issues such as false convergence due to communication delays, synchronization errors, and overall algorithm efficiency were addressed before moving to a full hardware testbed emulator [69].

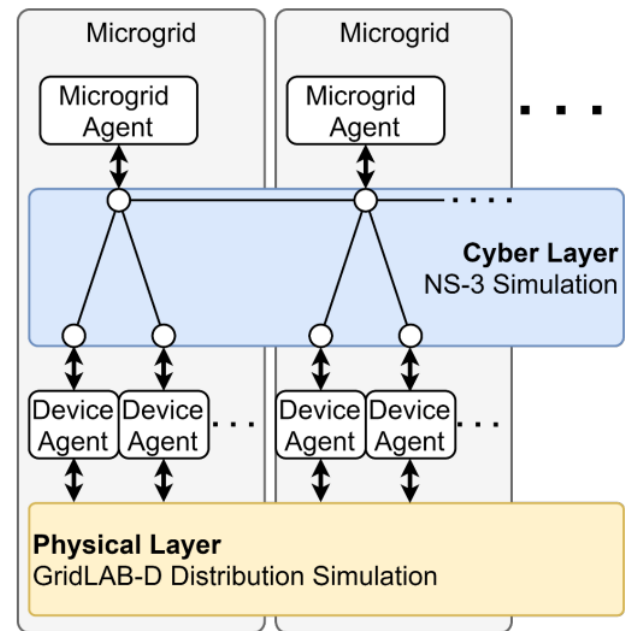


FIGURE 10: Federation architecture from [69] showing the connection between GridLAB-D modeling the power system, Python load device agents, the communication system modeling in ns-3, and Python microgrid agents.

##### 3) Autonomous Energy Systems

Autonomous energy systems (AES) use distributed hierarchical controls to manage very large numbers of DERs and other resources using nested cells to enable effective local management and reduced data needs for larger-scale coordination [70]. HELICS has been used extensively in evaluating and refining the algorithms and architecture for AES control schemes [71]. This includes both large-scale simulation of the entire San Francisco Bay area with millions of controllable DERs as well as detailed algorithmic development such

as evaluating communication delay constraints for primal-dual control schemes [72], [73]

#### 4) GridAPPS-D

GridAPPS-D is an advanced distribution system management platform largely developed and maintained by PNNL [74]. The platform allows the development and evaluation of various distribution management applications and evaluation of their impacts on locally managed simulation models. Co-simulation via HELICS is an essential tool in the platform as it allows the various controllers and applications to be dynamically deployed on the existing models. This degree of modularity gives GridAPPS-D flexibility and versatility to experiment and evaluate new distribution system management techniques and technologies. Figure 11 provides a high-level look at how GridAPPS-D is envisioned as a platform to facilitate an ecosystem of interoperable distribution system management software applications for utilities, solution providers, and researchers to support an advanced multi-stakeholder, multi-objective grid.

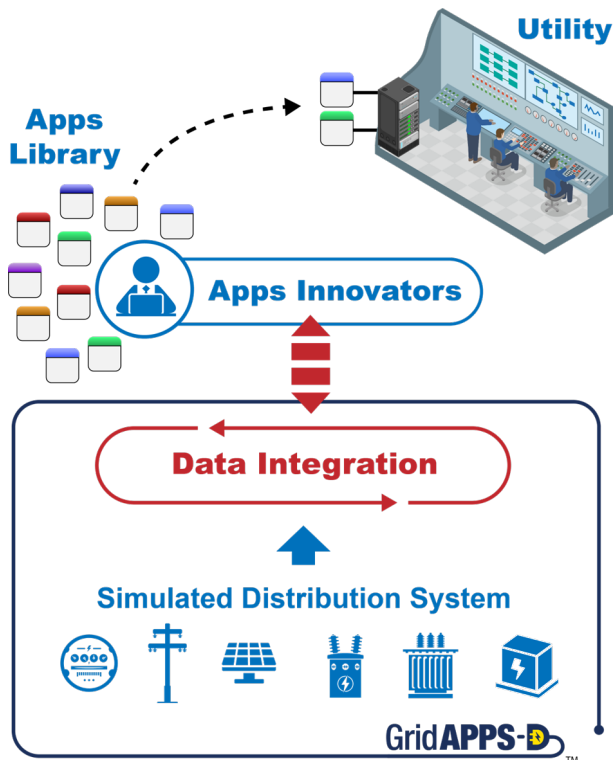


FIGURE 11: High-level overview of the architecture of GridAPPS-D [74] and its role as a platform to facilitate an ecosystem of interoperable distribution system management software applications for utilities, solution providers, and researchers.

#### 5) GO-Solar

The Grid Optimization with Solar (GO-Solar) project [75] developed a novel multi-part control approach that combines predictive state estimation—through matrix comple-

tion and multi-kernel learning—and on-line multi-objective optimization—using voltage-load sensitivity to guide a high-speed single-step gradient with linearized power flow. Together, these can efficiently manage extreme penetrations of solar and other DERs using only a few measurement points and only a few control nodes. HELICS was used to test this multi-part control scheme at scale as part of a large-scale integrated transmission-distribution operational co-simulation with more than 400 real-world feeders covering the entire island of Oahu. HELICS was also used to test the algorithms with more than 90 hardware devices through power hardware in the loop to explore the impacts of actual device behavior [52], [53].

#### 6) TESP

The Transactive Energy Simulation Platform [59] was developed as a means of allowing designers of transactive energy systems to more readily evaluate their performance. Such systems are generally multi-domain, frequently with modeling of the transmission and distribution systems both as power delivery networks but also as market operation environments. Detailed modeling of customer loads is often required to allow minute-by-minute control actions to be taken. TESP provides an integration of suitable tools for modeling transactive environments such as GridLAB-D for distribution systems, including distributed generation and residential thermodynamic modeling [76], PYPOWER [77] and AMES [78] for transmission systems, Energy+ for large commercial models [37], ns-3 for communication system modeling of control and coordination signals [79], and a collection of Python agents that manage transactive loads and implement portions of the transactive markets. TESP has been used to perform a number of analyses, including an initial demonstration of TESP capabilities [80] (see Figure 12 for an overview of the co-simulation data-exchange topology), a consensus-based fully-decentralized transactive mechanism [38], and the previously mentioned ERCOT-scale evaluation of an integrated retail and wholesale real-time and day-ahead energy market [39].

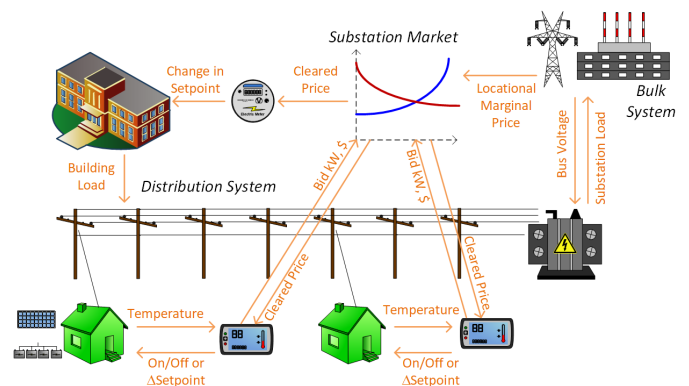


FIGURE 12: Overview of the co-simulation data-passing topology used in one of the initial use cases (see [80]) for PNNL's Transactive Energy Simulation Platform [59].

### E. CYBER-PHYSICAL: COMMUNICATIONS, SECURITY, AND MORE

The “smart grid” is fundamentally defined by the expansion of both communication and automation in the power system. Systems and components that were locally autonomous or completely disconnected from management systems of the power system are and have been connected, allowing for new power system management techniques. This newly expanded and connected network introduces new complexities and potential vulnerabilities; communication systems are not always reliable, distributed automation throughout the power system can fail or behave in unanticipated ways, and malicious actors have new avenues to compromise normal power system operation. With the growth in the smart grid over the past decade there has also come a need to perform new analysis the bridge the cyber and physical domains, and co-simulation is a primary tool for performing such analysis.

A number of efforts are underway to develop structures and systems to model cyber-physical interactions between the power grid and communication networks. For example, the Agile Co-simulation for Cyber Energy System Security (ACCESS) [81] seeks to understand cyber-related system impacts on infrastructure systems including gas and grid.

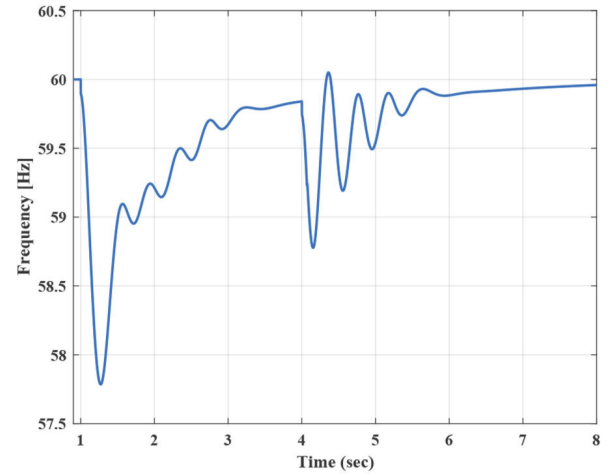
Other efforts take advantage of the various ways HELICS supports introducing communications into co-simulations. For example, the CPDS project [34], [63] described earlier uses HELICS’ built-in filters to introduce delays into the communication stream. In contrast, [82], [83] integrates a full packet-level communication simulation using ns-3 to evaluate the detailed trade-offs among various low-level communication protocols as part of a hybrid home/grid coordination control scheme.

#### 1) Microgrid Control

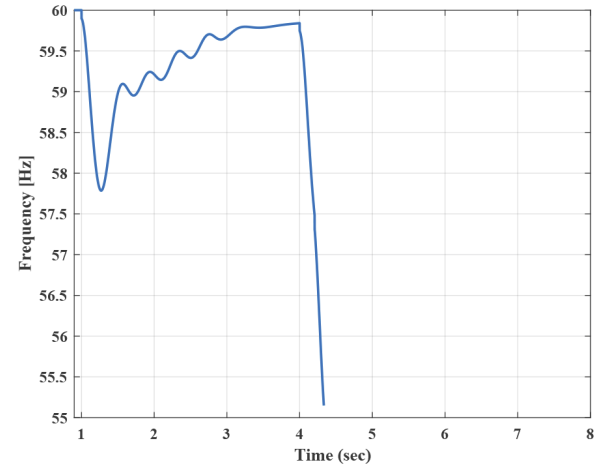
Microgrids as a resilience strategy (and more generally as a load and DER coordination strategy) have been studied for over a decade, and the growth of inverter-based resources (IBRs) has created both the possibility of viable islanded operation as well as complexity in coordinating local generation and load. Co-simulation provides a way of evaluating complex control and coordination mechanisms and strategies by linking high-fidelity power system models with controllers defined in Python or MATLAB.

In [41], a test framework for microgrid control is developed to demonstrate the adverse impact of non-ideal communications on the dynamic stability of networked microgrids connected to a centralized microgrid controller by using physical grid, communication, and control models (see Figure 13). [13] extends this further and evaluates the impact of different communication technologies, network structures, and media on the operations on an islanded microgrid using a battery energy storage system to offset the loss of variable generation. Ref. [38] implemented a consensus-based transactive system that required high-frequency communication between participants. To aid in the development of a system robust to communication system behavior, native HELICS

filters were used to replicate the effects of communication system delays and dropped packets (see Figure 14).



(a) Microgrid frequency under ideal communications



(b) Microgrid frequency under non-ideal communications

FIGURE 13: Results from [41] showing the impacts of frequency management in a microgrid when communication system impacts are considered.

#### 2) Wide Area Control

[84] as a part of NAERM (see Section IV-F) implemented a wide-area frequency controller with remote frequency measurements passing through a communication system simulator prior to reaching the controller. The analysis showed that as delays in the communication network increased the ability of a standard frequency controller to manage the system frequency failed, resulting in system separation.

### F. MULTI-INFRASTRUCTURE INTERACTIONS

The power system is steadily becoming more connected and reliant on resources outside of the direct control of utility operators, driven by consumer-owned DERs that provide resources at the edge of the grid, hundreds of generators that connect to a single interstate natural gas pipeline, and intense

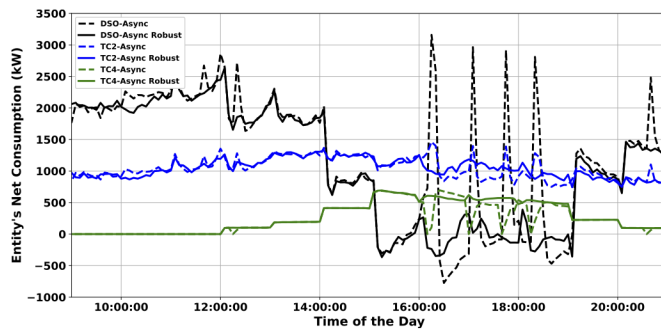


FIGURE 14: Results from [38] showing the difference in DER dispatch in a transactive system with controllers that are susceptible to non-ideal communications (dashed lines) and those that have been improved and are not (solid lines).

transportation loads that are mobile and less predictable. To continue to operate the power system on leaner reserve margins while simultaneously reducing costs and decarbonizing the system, holistic approaches to model, simulate, and understand these inter-dependencies are necessary. Co-simulation can help by bringing together existing simulators from various domains to capture these and other interactions.

The North American Energy Resilience Model (NAERM, [85]) is a DOE-led example, using co-simulation via HELICS to link simulators from different energy domains and modular data services to evaluate the resilience of the US energy system to extreme, national-scale events (*e.g.*, earthquakes, massive wildfires, polar vortex, etc.). NAERM combines commercial and open-source tools from power systems (steady state and dynamic) with production cost models, natural gas simulations, and communication system models to develop an engineering-class planning tool, capable of evaluating the interdependencies between these domains and addressing key resilience challenges.

Other efforts have focused on the interaction between the power grid and another specific infrastructure. For instance, [4] looks at the joint operations of the natural gas and grid systems, using both a small test system and a larger, more realistic model of the Belgium gas and grid systems. This work uniquely uses encoord's SAInt tool for both transient gas simulation and AC optimal power flow. This allows illustrating the gas-grid couplings. The very close match between the results of SAInt's native, single matrix gas-grid coupling and those where the gas and grid models are separated and coordinated through HELICS helps validate the effectiveness of co-simulation.

In the GEMINI-XFC project, HELICS is used to co-simulate grid and transportation systems [45]. Specifically, OpenDSS is used to simulate hundreds to thousands of distribution feeders representing the entire San Francisco Bay Area in conjunction with the BEAM transportation simulator that captures individual vehicle travel for the same footprint (see Figure 15 for the federation architecture). This platform then enables exploring various control schemes to manage

grid voltage and congestion with widespread EV and extreme fast charger use.

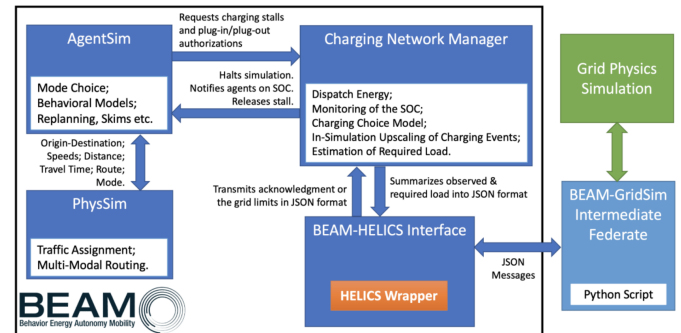


FIGURE 15: Federation architecture from [45] showing information passing between the transportation modeling in BEAM and the power system modeling in OpenDSS.

### G. REAL-TIME AND HARDWARE TESTING

While a majority of the effort has gone into software-based co-simulation, HELICS has been used in a number of circumstances to connect to real-time hardware devices. HELICS has a real-time mode that controls the time granted based on wall-clock timings. This can be used on one or all federates to drive the simulation. This allows HELICS to connect any software to a real-time hardware-in-the-loop (HIL) lab set including real-time simulators such as Opal-RT or RTDS, and other real-time systems including cybersecurity test beds.

For example, the signal driver for the hardware in the Skyfall [86] lab runs HELICS to connect with other simulation tools for power system simulation and communication.

HELICS also underpins the wide range of HIL testing conducted through the Advanced Distribution Management System (ADMS) Testbed [87]. This testbed uses HELICS to orchestrate interactions among commercial ADMS platform controls, large-scale distribution grid simulation in OpenDSS, and real-time controller and power HIL simulation with Opal-RT to connect dozens of DER hardware devices [88]. Hardware simulation with the ADMS testbed has led to over a dozen published papers, including evaluation of DERMS system performance [89], advanced control schemes such as data-enhanced hierarchical control (DEHC) [90], and next generation automation schemes [91]

### V. FUTURE WORK

The HELICS core library could be considered feature-complete as of HELICS v3.4 because it provides flexibility in defining the message-passing and timing interfaces necessary for integration of a variety of simulation tools and their corresponding software architectures. There are expected to be further developments to improve the usability of HELICS to allow new users an easier on-ramp into creating HELICS-based federations, debug the operation of existing federations, and more easily manage larger or more complex

federations. Prototype versions of some of these functionalities are present in the existing web interface, but further development effort is needed for them to reach a mature state. This is a significant challenge, because it is desirable that such a tool would be usable in all existing HELICS broker topologies when running in complex deployment strategies (e.g., single laptop, hybrid cloud, HPC, multi-institution) and on all supported OSes.

Relatedly, we anticipate that as HELICS gains wider adoption, there will be a need to provide or support some form of workflow and/or deployment tool. These tools provide ways of chaining together multiple existing tools to perform more complex analysis. For example, a workflow tool may make it possible to simulate the solar generation for a given geography and take that output and provide it as an input to a bulk power system tool. Sensitivity analysis and uncertainty quantification are also common analyses that greatly benefit from a workflow tool. Generally, we would like to see users be easily able to use a HELICS federation in these kinds of tools in the same way a single simulation tools would be used.

As of this writing, there has been somewhat limited testing to evaluate the performance of HELICS to handle large numbers of interfaces and/or interfaces that send large amounts of data, either as many small messages or individual large messages. Testing to date has not shown any problems, but more extensive evaluation would be helpful in validating the expected performance.

The new analysis needs that motivated the development of HELICS are migrating from academic concerns to real-world planning and operations concerns, motivating the development of co-simulation as part of industry practice. A key part of enabling these analysis is the integration of commercial industry software tools into the HELICS platform. As shown in Section III-D, some of these tools have been integrated, but many more are in common use throughout industry. Integration for some of these tools may require additional effort from the vendor to support one of the two integration methods discussed in Section III-A.

## REFERENCES

- [1] Gridwise Architecture Council, "GridWise Transactive Energy Framework Version 1.1," Gridwise Architecture Council, Tech. Rep., July 2019. [Online]. Available: [https://www.gridwiseac.org/pdfs/pnnl\\_22946\\_gwac\\_te\\_framework\\_july\\_2019\\_v1\\_1.pdf](https://www.gridwiseac.org/pdfs/pnnl_22946_gwac_te_framework_july_2019_v1_1.pdf)
- [2] FERC, "FERC Order 2222: Participation of Distributed Energy Resource Aggregations in Markets Operated by Regional Transmission Organizations and Independent System Operators," Sept 2020. [Online]. Available: [https://www.ferc.gov/sites/default/files/2020-09/E-1\\_0.pdf](https://www.ferc.gov/sites/default/files/2020-09/E-1_0.pdf)
- [3] Energy Information Administration, "Electric Power Annual 2018," United States Department of Energy, Washington, DC, Tech. Rep., Oct 2019. [Online]. Available: <https://www.eia.gov/electricity/annual/pdf/epa.pdf>
- [4] B. Sergi and K. Pambour, "An Evaluation of Co-Simulation for Modeling Coupled Natural Gas and Electricity Networks," *Energies*, vol. 15, no. 14, p. 5277, Jan. 2022, number: 14 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/1996-1073/15/14/5277>
- [5] NASPI Control Room Solutions Task Team, "Using Synchrophasor Data for Voltage Stability Assessment," North American Synchrophase Initiative, Richland, WA, Tech. Rep. NASPI-2015-TR-016, Oct 2015. [Online]. Available: <https://www.naspi.org/node/358>
- [6] D. Kosterev, J. Burns, N. Leitschuh, J. Anasis, A. Donahoo, D. Trudnowski, M. Donnelly, and J. Pierre, "Implementation and Operating Experience with Oscillation Detection Application at Bonneville Power Administration," in *2016 Grid of the Future Symposium*, 2016. [Online]. Available: <http://cigre-usnc.org/wp-content/uploads/2016/10/Kosterev.pdf>
- [7] IEEE, "IEEE Std 1516™-2010, IEEE Standard for Modeling and Simulation (M and S) High Level Architecture (HLA) Framework and Rules," Aug. 2010.
- [8] Modelica Association Project "FMI", "Functional Mockup Interface for Model Exchange and Co-Simulation, Version 2.0," Modelica Association Project "FMI", Tech. Rep., Jul. 2014.
- [9] S. Schütte and M. Sonnenschein, "Mosaik — Scalable Smart Grid Scenario Specification," in *Proceedings of the 2012 Winter Simulation Conference (WSC)*, Dec 2012, pp. 1–12.
- [10] B. Palmintier, E. Hale, T. M. Hansen, W. Jones, D. Biagioni, H. Sorensen, H. Wu, and B.-M. Hodge, "IGMS: An Integrated ISO-to-Appliance Scale Grid Modeling System," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1525–1534, Sep. 2016.
- [11] 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 *DEVS '14: Proceedings of the Symposium on Theory of Modeling & Simulation*. Society for Computer Simulation International, Apr. 2014, p. 36. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2665008.2665044>
- [12] B. Palmintier, D. Krishnamurthy, P. Top, S. Smith, J. Daily, and J. Fuller, "Design of the HELICS High-Performance Transmission-Distribution-Communication-Market Co-Simulation Framework," in *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, 2017, pp. 1–6. [Online]. Available: <http://ieeexplore.ieee.org/document/8064542/>
- [13] P. Thekkumparambath Mana, K. P. Schneider, W. Du, M. Mukherjee, T. Hardy, and F. K. Tuffner, "Study of Microgrid Resilience through Co-Simulation of Power System Dynamics and Communication Systems," *IEEE Transactions on Industrial Informatics*, December 2020.
- [14] J. Hansen, T. Hardy, and L. Marinovici, "Transactive Energy: Stabilizing Oscillations in Integrated Wholesale-Retail Energy Markets," in *2019 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb 2019, pp. 1–5. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8791658>
- [15] M. Mukherjee, L. Marinovici, T. Hardy, and J. Hansen, "Framework for large-scale implementation of wholesale-retail transactive control mechanism," *International Journal of Electrical Power and Energy Systems*, vol. 115, February 2020. [Online]. Available: <https://doi.org/10.1016/j.ijepes.2019.105464>
- [16] D. Rimorov, J. Huang, C. F. Mugombozi, T. Roudier, and I. Kamwa, "Power Coupling for Transient Stability and Electromagnetic Transient Collaborative Simulation of Power Grids," *IEEE Transactions on Power Systems*, pp. 1–1, 2021.
- [17] A. K. Bharati and V. Ajarapu, "A Scalable Multi-Timescale T D Co-Simulation Framework using HELICS," in *2021 IEEE Texas Power and Energy Conference (TPEC)*, Feb 2021, pp. 1–6.
- [18] B. Palmintier, E. Hale, T. Hansen, W. Jones, D. Biagioni, K. Baker, H. Wu, J. Giraldez, H. Sorensen, M. Lunacek, N. Merket, J. Jorgenson, and B.-M. Hodge, "Integrated Distribution-Transmission Analysis for Very High Penetration Solar PV (Final Technical Report)," National Renewable Energy Laboratory, Golden, CO, NREL Technical Report NREL/TP-5D00-65550, Jan. 2016, bibtext: palmintier.integrated\_2016. [Online]. Available: <http://www.nrel.gov/docs/fy16osti/65550.pdf>
- [19] M. M. Rezvani, S. Mehraeen, J. R. Ramamurthy, and T. Field, "Interaction of Transmission-Distribution System in the Presence of DER Units - Co-simulation Approach," *IEEE Open Journal of Industry Applications*, pp. 1–1, 2020.
- [20] N. Kang, R. Singh, J. T. Reilly, and N. Segal, "Impact of Distributed Energy Resources on the Bulk Electric System," Argonne National Laboratory, Chicago, IL, Tech. Rep. ANL/ESD-17/26, Nov. 2017. [Online]. Available: <https://anl.app.box.com/s/tknbyvu7xebsh98npscjd9k7b4nhsgp>
- [21] Battelle Memorial Institute, Lawrence Livermore National Security LLC, and Alliance for Sustainable EnergyLLC, "HELICS," 2018-2023. [Online]. Available: <https://github.com/GMLC-TDC/HELICS>
- [22] L. Barbierato, P. Rando Mazzarino, M. Montarolo, A. Macii, E. Patti, and L. Bottacioli, "A comparison study of co-simulation frameworks for multi-energy systems: the scalability problem," in *Proceedings of*

- the Energy Informatics Academy Conference 2022*, vol. 5, Dec 2022. [Online]. Available: <https://doi.org/10.1186/s42162-022-00231-6>
- [23] Battelle Memorial Institute, Lawrence Livermore National Security LLC, and Alliance for Sustainable Energy LLC, "PyHELICS," 2020-2023. [Online]. Available: <https://python.helics.org/>
- [24] —, "jELICS," 2022-2023. [Online]. Available: <https://github.com/GMLC-TDC/jHELICS>
- [25] —, "matHELICS," 2022-2023. [Online]. Available: <https://github.com/GMLC-TDC/matHELICS>
- [26] —, "HELICS.jl," 2019-2023. [Online]. Available: <https://github.com/GMLC-TDC/HELICS.jl>
- [27] —, "HELICS-FMI," 2021-2023. [Online]. Available: <https://github.com/GMLC-TDC/HELICS-FMI>
- [28] —, "HELICS-HLA," 2018-2019. [Online]. Available: <https://github.com/GMLC-TDC/HELICS-HLA>
- [29] R. Zimmerman, C. Murillo-Sánchez, and R. 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.
- [30] Pacific Northwest National Laboratory, "MATPOWER-wrapper," 2023. [Online]. Available: <https://github.com/GMLC-TDC/MATPOWER-wrapper>
- [31] N. Panossian, T. Elgindy, D. Wallison, and B. Palmintier, "Synthetic, Realistic Transmission and Distribution Co-Simulation for Voltage Control Benchmarking," in *Proceedings of the IEEE Texas Power and Energy Conference (TPEC)*, Virtual, Feb. 2021.
- [32] A. Latif and K. Duwadi, "PyPSSE," Aug. 2021. [Online]. Available: <https://github.com/NREL/PyPSSE>
- [33] H. Cui and F. Li, "ANDES: A Python-Based Cyber-Physical Power System Simulation Tool," in *2018 North American Power Symposium (NAPS)*, Sep. 2018, pp. 1–6.
- [34] W. Wang, X. Fang, and A. Florita, "Impact of DER Communication Delay in AGC: Cyber-Physical Dynamic Co-simulation," in *2021 IEEE 48th Photovoltaic Specialists Conference (PVSC)*, Jun. 2021, pp. 2616–2620, iSSN: 0160-8371.
- [35] W. Wang, M. Cai, X. Fang, and C. Irwin, "Impact of Open Communication Networks on Load Frequency Control with Plug-In Electric Vehicles By Cyber-Physical Dynamic Co-simulation," in *2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Jan. 2023, pp. 1–5, iSSN: 2472-8152.
- [36] W. Wang, X. Fang, and A. Florita, "Impact of DER Communication Delay in AGC: Cyber-Physical Dynamic Co-simulation," in *2021 IEEE 48th Photovoltaic Specialists Conference (PVSC)*, June 2021, pp. 2616–2620.
- [37] TESP Examples and Demonstrations - Energy+. [Online]. Available: <https://tesp.readthedocs.io/en/latest/demonstrations/energyplus.html>
- [38] M. Mukherjee, T. Hardy, J. C. Fuller, and A. Bose, "Implementing Multi-Settlement Decentralized Electricity Market Design for Transactive Communities with Imperfect Communication," *Applied Energy*, vol. 306, p. 117979, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306261921012824>
- [39] H. M. Reeve, A. Singhal, A. Tbaileh, R. G. Pratt, T. D. Hardy, J. D. Doty, L. D. Marinovici, S. R. Bender, M. A. Pelton, and M. R. Oster, "DSO+T: Integrated System Simulation DSO+T Study: Volume 2," Pacific Northwest National Laboratory, Richland, WA, Tech. Rep., January 2022. [Online]. Available: <https://www.osti.gov/biblio/1842488>
- [40] G. Kandaperumal, K. P. Schneider, and A. K. Srivastava, "A data-driven algorithm for enabling delay tolerance in resilient microgrid controls using dynamic mode decomposition," *IEEE Transactions on Smart Grid*, vol. 13, no. 4, pp. 2500–2510, 2022.
- [41] B. Bhattarai, L. Marinovici, M. Touhiduzzaman, F. K. Tuffner, K. P. Schneider, J. Xie, P. Thekkumparambath Mana, W. Du, and A. Fisher, "Studying Impacts of Communication System Performance on Dynamic Stability of Networked Microgrid," *IET Smart Grid*, vol. 3, no. 5, pp. 667–676, 2020.
- [42] B. Bhattarai, L. Marinovici, P. S. Sarker, and A. Orrell, "Miracle co-simulation platform for control and operation of distributed wind in microgrid," *IET smart grid*, vol. 5, no. 2, pp. 90–100, 2022.
- [43] A. Latif, "CYMEpy," Aug. 2021. [Online]. Available: <https://github.com/GMLC-TDC/cymepy>
- [44] A. Latif, D. Thomas, and K. Duwadi, "PyDSS," Aug. 2021. [Online]. Available: <https://github.com/NREL/PyDSS>
- [45] N. V. Panossian, H. Laarabi, K. Moffat, H. Chang, B. Palmintier, A. Meintz, T. E. Lipman, and R. A. Waraich, "Architecture for Co-Simulation of Transportation and Distribution Systems with Electric Vehicle Charging at Scale in the San Francisco Bay Area," *Energies*, vol. 16, no. 5, p. 2189, Jan. 2023, number: 5 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/1996-1073/16/5/2189>
- [46] N. Gray, R. Sadnan, A. Bose, and A. Dubey, "Effects of Communication Network Topology on Distributed Optimal Power Flow for Radial Distribution Networks," in *2021 North American Power Symposium (NAPS)*, Nov 2021, pp. 1–6.
- [47] W. Lardier, "ASGARDS-H: Enabling Advanced Smart Grid Cyber-Physical Attacks, Risk and Data Studies with HELICS," Master's thesis, Concordia University, Montreal, Canada, Nov 2020. [Online]. Available: [https://spectrum.library.concordia.ca/id/eprint/987690/1/Lardier\\_MASc\\_S2021.pdf](https://spectrum.library.concordia.ca/id/eprint/987690/1/Lardier_MASc_S2021.pdf)
- [48] "SAInt I Case Study: HELICS+ Natural Gas and Grid Validation and Optimization," 2022. [Online]. Available: <https://encord.com/CaseStudyHELICS.html>
- [49] E. Tataru, "NGTransient: A Hydraulic Pipeline and Optimal Control Model for Simulating Pipeline Disruptions and Impacts on Electric Power Generation," in *Resilience Week 2020*, Oct 2020. [Online]. Available: <https://doi.org/10.1186/s42162-022-00231-6>
- [50] T. Haines, B. M. Garcia, W. Vining, and M. Lave, "A Co-Simulation Approach to Modeling Electric Vehicle Impacts on Distribution Feeders During Resilience Events," in *2021 Resilience Week (RWS)*, Oct 2021, pp. 1–5.
- [51] F. K. Tuffner, J. Undrill, D. Schoffield, J. Eto, D. Kosterev, and R. Quint, "Distribution-Level Impacts of Plug-in Electric Vehicle Charging on the Transmission System during Fault Conditions," Pacific Northwest National Laboratory, Richland, WA, Tech. Rep. PNNL-31558, Oct 2021. [Online]. Available: <https://www.osti.gov/servlets/purl/1832905>
- [52] J. Wang, J. Simpson, R. Yang, B. Palmintier, S. Tiwari, and Y. Zhang, "Hardware-in-the-Loop Evaluation of an Advanced Distributed Energy Resource Management Algorithm," in *2021 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb. 2021, pp. 1–5, iSSN: 2472-8152.
- [53] —, "Performance Evaluation of an Advanced Distributed Energy Resource Management Algorithm," in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct. 2021, iSSN: 2472-8152.
- [54] D. Cutler, T. Kwasnik, S. Balamurugan, T. Elgindy, S. Swaminathan, J. Maguire, and D. Christensen, "Co-simulation of transactive energy markets: A framework for market testing and evaluation," *International Journal of Electrical Power & Energy Systems*, vol. 128, p. 106664, Jun. 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0142061520342095>
- [55] B. Palmintier, "How Use Cases Drove the Design of the HELICS Co-Simulation Framework," Online, Mar. 2018. [Online]. Available: <https://www.osti.gov/biblio/1669395-how-use-cases-drove-design-helics-co-simulation-framework>
- [56] H. Jain, B. Palmintier, D. Krishnamurthy, I. Krad, and E. Hale, "Evaluating the Impact of Price-Responsive Load on Power Systems Using Integrated T&D Simulation," in *Proc. of 2019 Integrated Smart Grid Technologies Conference*, Washington, DC, Feb. 2019.
- [57] E. Ela and M. O'Malley, "Studying the Variability and Uncertainty Impacts of Variable Generation at Multiple Timescales," *IEEE Transactions on Power Systems*, vol. 27, no. 3, pp. 1324–1333, Aug. 2012.
- [58] D. P. Chassin, J. C. Fuller, and N. Djilali, "GridLAB-D: An Agent-Based Simulation Framework for Smart Grids," *Journal of Applied Mathematics*, vol. 2014, pp. 1–12, 2014. [Online]. Available: <http://www.hindawi.com/journals/jam/2014/492320/>
- [59] Q. Huang, T. McDermott, Y. Tang, A. Makhmalbaf, D. Hammerstrom, A. Fisher, L. Marinovici, and T. D. Hardy, "Simulation-Based Valuation of Transactive Energy Systems," *Power Systems, IEEE Transactions on*, pp. 1–1, May 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8360969/>
- [60] S. Hanif, L. Marinovici, M. Pelton, T. Hardy, and T. E. McDermott, "Simplified Transactive Distribution Grids for Bulk Power System Mechanism Development," in *2021 IEEE Power & Energy Society General Meeting (PESGM)*, July 2021.
- [61] A. K. Bharati and V. Ajjarapu, "Smt-d co-simulation framework with helics for future-grid analysis and synthetic measurement-data generation," *IEEE Transactions on Industry Applications*, vol. 58, no. 1, pp. 131–141, Jan 2022.
- [62] A. K. Bharati, V. Ajjarapu, W. Du, and Y. Liu, "Role of distributed inverter-based-resources in bulk grid primary frequency response through helics

- based smtd co-simulation,” *IEEE Systems Journal*, vol. 17, no. 1, pp. 1071–1082, March 2023.
- [63] W. Wang, X. Fang, H. Cui, F. Li, Y. Liu, and T. J. Overbye, “Transmission-and-Distribution Dynamic Co-Simulation Framework for Distributed Energy Resource Frequency Response,” *IEEE Transactions on Smart Grid*, vol. 13, no. 1, pp. 482–495, Jan. 2022, conference Name: IEEE Transactions on Smart Grid.
- [64] S. Choi, H. Zhang, W. Tootle, Z. Flores, and F. Howell, “Real-Time SE Export Cases based WI Dynamic Simulation Update,” On-line, Nov. 2021. [Online]. Available: [https://www.wecc.org/\\_layouts/15/WopiFrame.aspx?sourcedoc=/Administrative/11b\\_Choi\\_Real-Time%20SE%20Export%20Cases%20based%20WI%20Dynamic%20Simulation%20Update\\_JSIS%202021.pdf&action=default&DefaultItemOpen=1](https://www.wecc.org/_layouts/15/WopiFrame.aspx?sourcedoc=/Administrative/11b_Choi_Real-Time%20SE%20Export%20Cases%20based%20WI%20Dynamic%20Simulation%20Update_JSIS%202021.pdf&action=default&DefaultItemOpen=1)
- [65] B. Bhattarai, G. S., P. Thekkumparambath Mana, and J. Fuller, “CleanStart DERMS: Modeling Effort Update,” Pacific Northwest National Laboratory, Richland, WA, Tech. Rep. PNNL-27644, June 2018. [Online]. Available: <https://www.osti.gov/biblio/1768022-cleanstart-derms-modeling-effort-update>
- [66] K. P. Schneider, S. Laval, B. Ollis, K. Prubaker, L. Tolbert, S. Essakiappan, R. Tucker, W. Du, J. Xie, L. D. Marinovici, B. P. Bhattarai, D. Lawrence, J. Hambrick, N. Shepard, M. Ferrarai, C. Murphy, Y. Velaga, Y. Liu, P. Kritprajun, Y. Liu, L. Zhu, Q. Dong, M. Manjrekar, and P. R. Chowdhury, “Duke-RDS Final Report,” Pacific Northwest National Lab. (PNNL), Richland, WA (United States), Tech. Rep. PNNL-32075, Sep. 2021. [Online]. Available: <https://www.osti.gov/biblio/1834057>
- [67] A. Narayanan and T. Hardy, “Synthetic data generation for machine learning model training for energy theft scenarios using cosimulation,” *IET Generation, Transmission & Distribution*, vol. 17, no. 5, pp. 1035–1046, 2023. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/gtd2.12619>
- [68] J. Coignard, T. Nouidui, C. Gehbauer, M. Wetter, J.-Y. Joo, P. Top, R. R. Soto, B. Kelley, and E. Stewart, “Cyder - a co-simulation platform for grid analysis and planning for high penetration of distributed energy resources,” in *2018 IEEE Power & Energy Society General Meeting (PESGM)*, 2018, pp. 1–5.
- [69] K. P. Schneider, J. Glass, C. Klauber, B. Ollis, M. Reno, A. Dubey, M. Burck, J. Xie, L. T. Wall, L. D. Marinovici, T. L. Vu, W. Du, D. Nurdy, A. Wiley, T. Mannon, D. Holman, N. K. Smith, W. Dawson, N. Shepard, M. Ferrari Maglia, J. Hernandez-Alvidrez, N. S. Gurule, A. Bose, and J. Hambrick, “Citadels final report (gmlc 2.2.1: Citadels),” Pacific Northwest National Laboratory, Richland, USA, Tech. Rep. PNNL-33889, 8 2023. [Online]. Available: <https://www.osti.gov/biblio/1995452>
- [70] B. Kroposki, A. Bernstein, J. King, D. Vaidhynathan, X. Zhou, C.-Y. Chang, and E. Dall’Anese, “Autonomous Energy Grids: Controlling the Future Grid With Large Amounts of Distributed Energy Resources,” *IEEE Power and Energy Magazine*, vol. 18, no. 6, pp. 37–46, 2020, publisher: IEEE.
- [71] D. Vaidhynathan and J. King, “Autonomous Energy System Simulation Capabilities – Ultra-Large Scale DER Deployment,” Golden, CO, Aug. 2020. [Online]. Available: <https://www.nrel.gov/grid/assets/pdfs/aes-king.pdf>
- [72] J. Comden, J. Wang, and A. Bernstein, “Study of Communication Boundaries of Primal-Dual-Based Distributed Energy Resource Management Systems (DERMS),” in *2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Jan. 2023, pp. 1–5, iSSN: 2472-8152.
- [73] —, “Evaluation of Communication Issues in Primal-Dual-Based Distributed Energy Resource Management Systems (DERMS),” in *IEEE Power Engineering Society General Meeting, 2023*. Orlando, FL: IEEE, Jul. 2023, arXiv:2305.03854 [math]. [Online]. Available: <http://arxiv.org/abs/2305.03854>
- [74] R. B. Melton, K. P. Schneider, E. Lightner, T. E. McDermott, P. Sharma, Y. Zhang, F. Ding, S. Vadari, R. Podmore, A. Dubey, R. W. Wies, and E. G. Stephan, “Leveraging standards to create an open platform for the development of advanced distribution applications,” *IEEE Access*, vol. 6, pp. 37 361–37 370, 2018.
- [75] R. Yang, Y. Zhang, B. Palmintier, X. Zhu, Y. Liu, A. Bernstein, J. Simpson, W. Wang, I. Krad, M. Martin, M. Emmanuel, J. Wang, M. Asano, A. Hirayama, and W.-H. Chen, “Grid optimization with solar (GO-Solar),” Feb. 2022. [Online]. Available: <https://www.nrel.gov/docs/fy22osti/82026.pdf>
- [76] D. P. Chassin, K. P. Schneider, and C. Gerkensmeyer, “GridLAB-D: An open-source power systems modeling and simulation environment,” in *IEEE PES Transmission and Distribution Conference and Exposition*. IEEE, May 2008, pp. 1–5. [Online]. Available: <http://ieeexplore.ieee.org/document/4517260/>
- [77] PYPPOWER Github repository. [Online]. Available: <https://github.com/rwl/PYPPOWER>
- [78] H. Li and L. Tesfatsion, “The ames wholesale power market test bed: A computational laboratory for research, teaching, and training,” in *2009 IEEE Power & Energy Society General Meeting*, 2009, pp. 1–8.
- [79] ns-3 network simulator. [Online]. Available: <https://nsnam.org>
- [80] S. E. Widergren, D. J. Hammerstrom, Q. Huang, K. Kalsi, J. Lian, A. Makhmalbaf, T. E. McDermott, D. Sivaraman, Y. Tang, A. Veeramany, and J. C. Woodward, “Transactive Systems Simulation and Valuation Platform Trial Analysis,” Pacific Northwest National Laboratory (PNNL), Richland, WA (United States), Richland, WA, Tech. Rep. PNNL-26409, Apr. 2017. [Online]. Available: <http://www.osti.gov/servlets/purl/1379448/>
- [81] B. M. Kelley, H. Scott, C. C. Sun, and N. Venethongkham, “Energy resilience for mission assurance: Agile co-simulation for cyber energy system security (access), model advancements for resilience analysis,” 8 2022. [Online]. Available: <https://www.osti.gov/biblio/1883445>
- [82] J. Zhang, S. M. S. Alam, A. R. Florita, A. Hasandka, J. Wei-Kocsis, D. Wang, L. Yang, and B. S. Hodge, “Opportunistic Hybrid Communications Systems for Distributed PV Coordination,” National Renewable Energy Lab. (NREL), Golden, CO (United States), Tech. Rep. NREL/TP-5D00-73716, Mar. 2020. [Online]. Available: <https://www.osti.gov/biblio/1606149>
- [83] J. Zhang, J. Daily, R. A. Mast, B. Palmintier, D. Krishnamurthy, T. Elgindy, A. Florita, and B.-M. Hodge, “Development of HELICS-based High-Performance Cyber-Physical Co-simulation Framework for Distributed Energy Resources Applications,” in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Nov. 2020, pp. 1–5.
- [84] Pacific Northwest National Laboratory, “Wide Area Control HELICS Co-simulation,” 2019. [Online]. Available: <https://github.com/GMLC-TDC/HELICS-Use-Cases/tree/master/PNNL-Wide-Area-Control>
- [85] US Department of Energy Office of Electricity, “North American Energy Resilience Model,” 2019. [Online]. Available: [https://www.energy.gov/sites/prod/files/2019/07/f65/NAERM\\_Report\\_public\\_version\\_072219\\_508.pdf](https://www.energy.gov/sites/prod/files/2019/07/f65/NAERM_Report_public_version_072219_508.pdf)
- [86] J. Helms, “Defending the Vulnerable Power Grid,” Dec. 2018. [Online]. Available: <https://str.llnl.gov/2018-12/helmsE>
- [87] A. Pratt, M. Baggu, F. Ding, S. Veda, I. Mendoza, and E. Lightner, “A test bed to evaluate advanced distribution management systems for modern power systems,” in *IEEE EUROCON 2019-18th International Conference on Smart Technologies*. IEEE, 2019, pp. 1–6.
- [88] J. Wang, H. Padullaparti, F. Ding, M. Baggu, and M. Symko-Davies, “Voltage Regulation Performance Evaluation of Distributed Energy Resource Management via Advanced Hardware-in-the-Loop Simulation,” *Energies*, vol. 14, no. 20, p. 6734, Jan. 2021, number: 20 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/1996-1073/14/20/6734>
- [89] J. Wang, M. Blonsky, F. Ding, S. C. Drew, H. Padullaparti, S. Ghosh, I. Mendoza, S. Tiwari, J. E. Martinez, J. J. D. Dahdah, F. A. M. Bazzani, M. Baggu, M. Symko-Davies, C. Bilby, and B. Hannegan, “Performance Evaluation of Distributed Energy Resource Management via Advanced Hardware-in-the-Loop Simulation,” in *2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb. 2020, pp. 1–5, iSSN: 2472-8152.
- [90] H. Padullaparti, J. Wang, S. Veda, M. Baggu, and A. Golnas, “Evaluation of Data-Enhanced Hierarchical Control for Distribution Feeders With High PV Penetration,” *IEEE Access*, vol. 10, pp. 42 860–42 872, 2022, conference Name: IEEE Access.
- [91] J. Wang, H. Padullaparti, S. Veda, I. Mendoza, S. Tiwari, and M. Baggu, “Performance Evaluation of Next-Generation Grid Automation and Controls with High PV Penetration,” in *2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Jan. 2023, pp. 1–5, iSSN: 2472-8152.



Laboratory, he has focused his research efforts on distribution system power analysis, co-simulation using HELICS, and transactive energy system design and analysis.

TREVOR D. HARDY (S'97-SM'22) received his B.S. degree in engineering from LeTourneau University in Longview Texas, USA, and his Ph. D. in electrical engineering from Wichita State University in Wichita, Kansas, USA. He has worked for Micron Technology in Boise, ID as a product engineer doing DRAM testing for new process technologies and as an embedded systems designer for Cessna Aircraft Company in Wichita, Kansas. Since joining Pacific Northwest National



for DOE's NAERM efforts. Mr. Fuller is a senior member in IEEE, acting as past-Chair of the Distribution System Analysis Subcommittee and Chair of the Test Feeder Working Group.

JASON C. FULLER (S'08-M'10-SM'18) received his B.S. degree in Physics from the University of Washington and his M.S. degree in Electric Engineering from Washington State University. He is currently a Technical Group Leader and Research Engineer at Pacific Northwest National Laboratory. Project highlights include the development and application of GridLAB-D and related co-simulation tools, such as FNCS and HELICS. He is currently the Infrastructure Modeling Lead



Solutions, Santa Clara University, NOLS, and the Naval Center for Space Technology. His current research focuses on co-optimized T&D design and effective integration of distributed energy resources into low-carbon multi-energy systems across technical, social, economic, regulatory, and equity dimensions.

BRYAN PALMINTIER (M'04-SM'15) holds a Ph.D. in Engineering Systems from MIT; a Degree of Engineer in mechanical engineering and an M.S. in aero/astro engineering from Stanford; and a B.S. in aerospace engineering from Georgia Tech. He is currently the Group Manager for the Transmission & Distribution Interactions (T&D) group and a principal research engineer in the Grid Planning and Analysis Center at NREL. His work prior to NREL included time with RMI, Energy



more on the power grid include development of open source tools for power systems (including GridDyn, a power system simulator, and HELICS), grid sensors and data analytics, and hardware-in-the-loop testing.

PHILIP L. TOP (S'98-M'07) received his B.S. degree in Engineering from Dordt University in Sioux Center, IA and his M.S. in Engineering and Ph.D in Electrical Engineering from Purdue University. He is currently a Group Leader and Research Engineer at Lawrence Livermore National Laboratory. Research highlights include work on different remote sensing technologies including ultrawideband radar, X-ray, acoustics, LIDAR, and satellite imagery. Recent work has focused



developing Canada's long-term supply and demand energy and greenhouse gas emissions projections model.

DHEEPAK KRISHNAMURTHY (S'2012-M'2015) received his B.E. in Electrical and Electronics Engineering from SSN University and M.S. degree in Electric Engineering from Iowa State University. He is currently a Senior Policy Advisor at Environment and Climate Change Canada. He has worked on the development of a number of energy system modeling and simulation software such as HELICS, PowerSimulations.jl, OpenDSS-Direct.py, AMES, etc. He is currently involved in