Software and Communications Platform for Simulation Environment of Complex Energy System (SimCES)

Petri Kannisto, Otto Hylli, Ville Heikkilä, Antti Supponen, Timo Aaltonen, Sami Repo, Kari Systä
Tampere University
Tampere, Finland
ORCID: 0000-0002-0613-8639

Antti Keski-Koukkari, Amir Safdarian, Anna Kulmala
VTT Technical Research Centre of Finland
Espoo, Finland
ORCID: 0000-0003-0490-9003

***Abstract*—Electricity networks are becoming more complex due to the integration of renewable and distributed energy resources, which necessitates tools to simulate more advanced management methods and multi-stakeholder interactions. To facilitate simulation, this article introduces a distributed communications platform. Unlike earlier platforms, this applies loose coupling and abstract software interfaces, enabling the users to flexibly select and parametrize simulation components. Furthermore, the components can operate on any software runtime. The components communicate via a RabbitMQ message bus, which provides asynchronous publish-subscribe communication. The work is in progress but has a functional prototype, enabling the simulation of an electricity grid and distributed energy resources. The prototypes under development will simulate the management of energy communities and electricity grids including the consideration of energy pricing, demand and weather conditions. The study contributes to the development of zero-emission energy systems and flexibility markets for grid operators and flexibility service providers.**

***Index Terms*—Message-oriented middleware, Power system simulation, Smart grids, Software architecture, Web services**

I. Introduction

The complexity of electricity distribution system has been steadily growing in recent years. This growth stems from an increase in uncertain, intermittent, and distributed generation, the penetration of electric vehicles and heat pumps, and the recent inclusion of Citizen Energy Communities and Renewable Energy Communities [1] in European legislation. Additionally, multiple stakeholders with sometimes conflicting goals are present in energy ecosystem. This complexity imposes new challenges in system-level validation and impact evaluation, and thus, generates a need for an evaluation platform where the joint effect of the energy domain actors can be evaluated, for both industrial and research purposes.

This paper was financially supported by the European project H2020 'INTERRFACE' (grant agreement number 824330) and national project 'ProCemPlus' funded by Business Finland (diaries 842/31/2019 and 8211/31/2018). The authors want to express their sincere gratitude.

978-1-6654-3597-0/21/$31.00 ©2021 IEEE

This article introduces *Simulation Environment of Complex Energy System* (SimCES), a platform to enable the distribution of simulation tasks to components that interact over a communication network. Here, 'complex' refers to a multi-stakeholder, multi-market environment. The goal is to enable the comparison of scenarios where not only power system actors and their properties vary but also where certain actors enforce control or provide instructions to other actors. This would facilitate the development of control algorithms, coordination schemes, and market-based flexibility provision for all stakeholders. To facilitate the customization of simulations, it should be straightforward to add and remove simulation components as well as parameterize these. Despite the current focus on non-real-time simulation, the long-term vision is the control of physical electricity networks as Hardware-in-the-Loop. SimCES can be considered to enable *co-simulation*, although the concept often refers to simulation in one computer.

The research question is:

> What kind of software platform enables the simulation of energy communities and electric grids to be distributed over a communication network, (1) enabling customizability in adding, removing and modifying components to simulate varying energy system scenarios, (2) enabling the simulation of cases where the actors have conflicting goals and operate against the global optimum, and (3) facilitating the development of control algorithms and coordination schemes?

Part (1) of the research question has a prototype. Parts (2) and (3) can build upon it but are currently under development.

Next, Section II reviews the related research. Then, Section III describes the design of SimCES, followed by a proof of concept in Section IV. Finally, Section V discusses the platform design and Section VI concludes the paper.

II. Related Work

Previous research has delivered publications about co-simulation platforms and interfaces. This section reviews the most relevant solutions focusing on energy systems.

To support the distribution of simulation tasks to multiple modules, Functional Mockup Interface (FMI [2]) specifies an interface. For instance, FMI has been applied to smart grids by Divshali et al. [3]. However, FMI lacks an interface for network communication and is thus unsuitable for SimCES.

Grigull et al. [4] have developed a distributed simulator system for electric grids. Similar to SimCES, the system enables distribution over a communication network. However, the communication protocols are point-to-point and require a Virtual Private Network (VPN) to connect laboratories. This makes the system tightly coupled and heavyweight compared to SimCES that is loosely coupled yet distributed and secure.

Kulmala et al. [5] proposed an information exchange platform for the utilization of distributed energy resources (DER). The platform enables distribution over Internet with commonly agreed interfaces but focuses on flexibility and DERs rather than aiming at a re-usable software architecture as SimCES.

Zhao et al. [6] suggested the co-simulation platform 'LICPIE' that utilizes a middleware to enable the distribution of simulation modules over a network. Compared to this study, LICPIE is more complex with its multiple layers and more oriented towards connecting existing simulation platforms rather than providing a single channel for interoperability. Furthermore, LICPIE lacks the publish-subscribe aspect in communication, which would enable loose coupling and better scalability if same data are streamed to multiple recipients.

SimCES is based on an AMQP 0-9-1 message bus similar to 'COCOP' architecture [7] that enables distributed simulation. However, COCOP focuses on the coordination in industrial process plants rather than energy systems.

Mosaik is a platform to enable distributed simulation [8]. Unlike this study, Mosaik is centralized and utilizes web sockets in communication, which makes it tightly coupled.

To conclude related work, no previous study enables the following combination: distribution of sub-simulators over a network, interfaces that do not depend on a specific product, and loose coupling to enable flexibility in changing the simulator network. That is, this study aims to fill a research gap.

III. Simulation Platform and Components

This section introduces the design of SimCES. It starts from high-level principles, followed by more detailed aspects.

A. Core Principles

The simulation platform has multiple objectives to fulfill, which necessitates careful design. The platform should make it straightforward to add, remove, and modify the components that participate in co-simulation. Furthermore, the components should communicate over a network that enables geographical distribution. Preferably, the platform should not enforce any particular product or software runtime, because this would unnecessarily limit the options of development. As the users develop methods and algorithms, they should be able to focus on a particular problem rather having to master the overall execution of the system.

To fulfill the requirements, SimCES follows Service-oriented Architecture or SOA [9, p. 291]. Particularly the following SOA principles apply in the design of components:

- *Loose coupling*: the interaction of components only necessitates logical rather than physical dependencies
- *Abstraction*: in each component, only expose the information necessary for interaction
- *Autonomy* (at design time): hide implementation details
- *Reusability*: the component is usable by multiple other components and in multiple scenarios
- *Composability*: the component can be orchestrated into an entity formed by multiple components

B. Message Bus and Topics

SimCES implements SOA principles with the *message bus* paradigm. This decouples components and runtimes from each other, enabling the components to be added, removed, or modified without affecting the others. A message bus covers both a communication medium and an information model, which sets requirements to the further platform design [10].

To realize loose coupling, SimCES components communicate with *topics* and *message queues*, which is an approach for publish-subscribe communication. That is, any component that generates information will send this to a dedicated topic identified by a string. Respectively, any component willing to receive this information subscribes for the topic. To receive messages from the topic, the component creates a message queue. The message bus hosts both the queues and topics. If any topic has multiple subscribers, the message bus will create a copy of published messages into the queue of each. The queue-based approach realizes asynchronism, because no subscriber can block the execution of others even if there were a delay in reading the messages. This differs from request-response communication (e.g., Restful services), where each client must query for data and therefore know the exact address of the service. Besides, request-response is synchronous. Compared to request-response, publish-subscribe enables loose coupling in 'time, space, and synchronization' [11].

As the communication protocol for the message bus, SimCES uses Advanced Message Queuing Protocol (AMQP) version 0-9-1. In AMQP 0-9-1, topic-based communication is only one of the supported communication schemes but satisfies the needs of SimCES. An advantage of AMQP is that it enables the geographical distribution of simulation components via Internet. For the AMQP implementation, SimCES uses RabbitMQ [12]. This is open source and available at no cost.

C. Execution and Timing Control

In distributed simulation, all components must have a common perception of time, which is realized with 'epochs' in SimCES. An epoch represents a slot of simulated time that enables the synchronization of otherwise asynchronous simulation. An epoch can represent, for example, the time between 1:00 and 2:00 p.m. on a certain day. When an epoch starts, each component has the permission to start operation, and the epoch can only end once all components have finished.

The epoch length can be, e.g., 1 hour or 15 minutes depending on the desired resolution, and this does not have to be constant.

An epoch starts when an 'epoch message' is published to the network, and all components receive this. This message comes from a component called Simulation Manager that controls the timing of epochs.

Within an epoch, the operation of a component can be triggered by either an epoch message or the output of another component. An epoch message can trigger a component that does not need any input from others. For example, some components simply publish values from timeseries to fabricate a real-life phenomenon, such as a component that submits a solar radiation value from a pre-generated timeseries. On the other hand, the radiation value can trigger another component to simulate the active power output of a solar plant, which will again be published to the message bus. It is notable although there are connections, these use publish-subscribe instead of a sequential point-to-point scheme.

An epoch can only end after all components have finished their calculation for it. After finishing, each component publishes a 'ready' message. Once Simulation Manager has received 'ready' from all components, it can start another epoch. If needed, the components can even iterate calculation together within one epoch. That is, a component that has already provided output would refresh based on the output of another. However, this necessitates rules in interaction to ensure convergence and a consensus whether the iteration has ended. Alternatively, iterative calculations can be encapsulated within a component so that iteration occurs internally. For example, the voltage dependency of an electrical load in a load flow calculation can be incorporated to the load flow component instead of two or more components. The decision on how to incorporate or divide iterative calculations is not trivial and needs to be decided case specifically.

Messaging is illustrated in Fig. 1, which contains example components C1-C4. Each of these receives the epoch message from Simulation Manager when an epoch starts and sends a ready message when finished. In between, the components send their output to the bus. To calculate, C2 needs the output of C1, and C4 needs output from C2 and C3. The output of C4 goes to one or more other components (i.e., C{n}).

Error handling is necessary due to the complexity of simulations and beneficial regarding the development of algorithms. For instance, an algorithm may fail to generate output due to non-convergence. It is important to detect this to enable the developers to improve the algorithm. Therefore, SimCES specifies that whenever a component has faced an unrecoverable error, it will signal this to Simulation Manager, which will end the simulation by signaling other components. Then, it is up to the human user to look at the situation.

D. Components

In this study, a *component* refers to a building block of simulation. Some components have a real-life counterpart, such as a 'load' or 'generator'. The components that represent real-life resources are called *energy domain components*. On the other hand, any components that only exist for simulation are called *platform components*. Fig. 2 illustrates the components that either exist or are being developed for SimCES. These are explained in the following paragraphs. In the future, new components can be developed as needed.

The platform components include Simulation Manager, Platform Manager and Logging System. Simulation Manager coordinates the execution of simulation run with epochs. Platform Manager coordinates the startup of each simulation run, as the Simulation Manager is instantiated for each run. Platform Manager is configurable to enable the inclusion and exclusion of components as well as parametrization, as this will enable the comparison of different energy scenarios. Logging System is important for the evaluation of simulation results. It consists of two components: Log Writer and Log Reader. Log Writer stores all messages delivered in a simulation run. Log Reader provides a Restful API (Application Programming Interface) to retrieve messages later. It can generate timeseries from messages, which is beneficial in data analysis.

The currently specified energy domain components are Load, Generator and Storage. The real-life correspondent of Load is whatever device or customer (or a group of these) that consumes electricity, such as a heater or stove. Generator is the logical opposite of Load, generating electricity, such as a solar or wind power plant or farm. Finally, a Storage stores energy. It can be a battery or a buffer of electric energy, such as a cooling or heating system.

The rest of components serve a variety of purposes. State Monitoring monitors the real-time state of an electric grid, whereas Grid is a simulation model of the grid. Economic Dispatch controls resources in an energy community connected

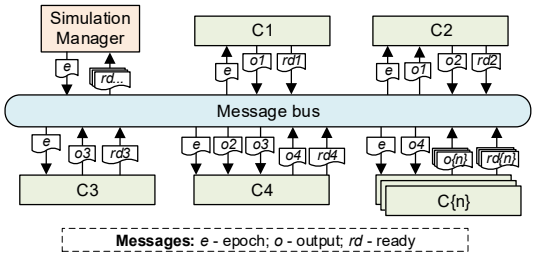

Fig. 1. Communication of epochs, calculated output and being ready

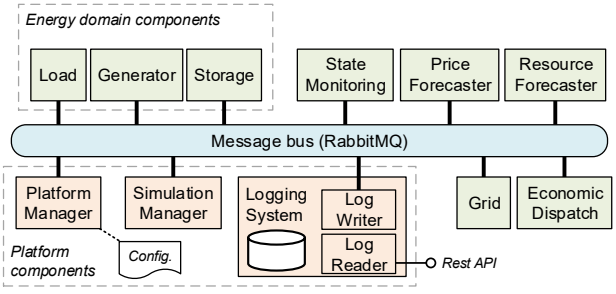

Fig. 2. Message bus and the components specified this far

to a network. It aims to reach advantage by applying flexibility where and when has the highest value, covering internal portfolio optimization when operating on day-ahead electricity market and several other markets. Finally, Price Forecast and Resource Forecast provide future information that Economic Dispatch can use in its control decisions.

E. Information Model

To enable communication with messages, there must be an information model. In SimCES, this specifies the content and structure of each message published by the components. First, this covers domain-specific information exchange. For instance, Load and Generator report their state with a *ResourceState* message, whereas Grid publishes output with *NetworkState*. Second, the information model enables the control of execution with messages, such as 'epoch' and 'ready'.

The messages share some structures, so these are modelled as reusable parts. For example, all messages are based on *AbstractMessage* that encloses fields, such as epoch number, the timestamp of message generation (in real time), and the identifier of the source component. The other reusable structures include quantities (value and unit of measure) as well as timeseries that appear in multiple messages.

The messages are serialized as JavaScript Object Notation (JSON). This is platform-independent and widely supported in development tools.

In the future, the information model could be aligned with standards. However, these are many and lack harmonization. The candidates include at least IEC 61850 to integrate field devices and IEC Common Information Model (CIM [13]) to describe resources and networks.

F. Deployment

Because SimCES aims to ease simulation, virtualization is applied in deployment to decrease manual work. In a conventional approach, each component would be deployed manually to a server. However, SimCES uses Docker [14], which enables applications to be set up virtually with a few lines of code. The applications are run in 'containers' that provide a runtime similar to a physical computer. The containers can be even grouped ('Docker Compose'). Such a group can be started and stopped with a single command, which is easy compared to manual deployment.

However, not all software can be easily virtualized, which necessitates manual deployment in some cases. This applies to tasks that perform processor- or memory-intensive calculation and may require even a server cluster. On the other hand, certain environments, such as Windows, are challenging to set up in Docker. Therefore, the related software is deployed manually. An example of this is the simulation of the Distribution System Simulator applied in Grid component.

G. Security

The RabbitMQ message bus supports basic security features, including user authentication, connection encryption and access control. These are disabled in the early development, but enabling is straightforward. In RabbitMQ, connections can be encrypted with TLS (Transport Layer Security). If needed, it is possible to develop a mechanism to encrypt even message contents for finer-grained confidentiality.

The component Log Reader has a Rest API to retrieve logged messages, which needs security. This should apply user authentication and traffic encryption, but this is future work.

If sensitive data are included in simulations, the future versions may require more security features. For instance, if customer data are included, only specific persons can have the access. Additionally, this would require logs to track if any unauthorized instances have accessed the data. As SimCES is developed further, the extent of the security measures must be considered according to the needs.

IV. Prototypes and Scenarios

To illustrate how the platform works and what kinds of simulations it enables, this section explains what has been implemented and what is planned for the future. First, this section describes the common simulator components. This is followed by a use case about a grid that hosts prosumers. Finally, the planned use cases are explained in short.

A. Common Components

All use cases require the base simulation platform components: Platform Manager, Simulation Manager, Log Writer, and Log Reader. Before a simulation can be started with the Platform Manager, there must be a running instance of RabbitMQ for communication and MongoDB [15] for Log Writer to store all messages published in the simulation.

Each simulation run is started with the Platform Manager by giving it a configuration file. This lists the components participating in the simulation and the parameters required by each component. The Platform Manager uses Docker to instantiate some components including Simulation Manager and Log Writer. To provide parameters to the components not deployed with Docker but manually, Platform Manager communicates via the message bus.

After the simulation run has been completed, it is time to analyze the results. This is enabled with the Log Reader that serves logged messages from MongoDB. Furthermore, it can generate timeseries from multiple messages.

B. Grid Scenario

The grid scenario simulates the behavior of a rural energy community: energy consumption and production. It includes the simulation of the electricity grid and a component to monitor the grid state. The scenario includes four houses without any controllable loads or storages. Two houses have a photovoltaic (PV) system, whereas one house has an electric vehicle (EV) being charged. The scenario spans one day and uses one-hour resolution, that is, a total of 24 epochs that simulate one hour. The network contains a total of 12 buses. Since none of the resources can be controlled, this scenario provides a reference point to future scenarios where some control actions can be issued to the resources of the network.

This scenario consists of the following components: Grid (a simulator using OpenDSS), State Monitoring, and seven Energy Domain Components that represent resources. The latter include five Loads (the four houses and the EV) and two Generators (the PV systems). Grid and State Monitoring are manually deployed components, whereas the others are deployed virtually with Docker. The main programming language of State Monitoring is Matlab, while the other components have been programmed in Python.

In this prototype, the behavior of each resource is predetermined and defined by a timeseries defining real and reactive power for each resource for each hour. The simulation-specific parameters, for example the grid model name, are given to the Grid and State Monitoring through the message sent by Platform Manager. When instantiated by the Platform Manager, each Energy Domain Component is given the resource name, type (Load or Generator) and a Comma-separated Values (CSV) file containing the power timeseries.

Fig. 3 illustrates the messaging sequence of the Grid scenario during one epoch. First, each Energy Domain Component (1..n) reports its state, i.e., power values. In addition, this message includes metadata, such as the source, simulation run and epoch (see Fig. 4). The state messages are delivered to Grid (subscribed for resource states) and Log Writer (subscribed for all messages to store them for later analysis). Once Grid has received all messages, it determines the state of the network and submits this to the message bus. This message is delivered to Log Writer and State Monitoring. Finally, State Monitoring determines the network status. If the status is not acceptable (e.g., congestion-related constraints are violated), State Monitoring will report the status to the message bus. Any subscribers will receive this message although only Log Writer in this scenario. It is notable that because all the communication occurs using the publish-subscribe pattern, the original message senders do not care about the receivers. Instead, the message bus delivers the messages based on topics that exist for each message type. This realizes loose coupling, which facilitates modifications and extensions in the system.

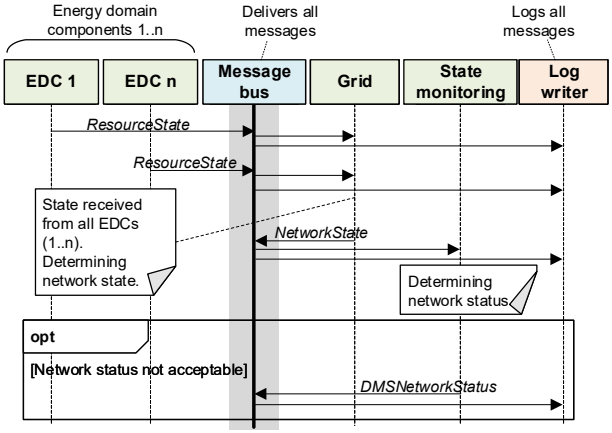

Fig. 3. Message sequence of the Grid scenario during one epoch omitting the epoch and ready messages.

Once simulated, it is possible to analyze the results with Log Reader. This enables the user to inspect how Grid determines network state based on resource states as well as how State Monitoring operates based on the information from Grid. This enables the testing and comparison of algorithms.

C. Scenarios Being Planned

The following simulations are being developed.

1) Energy Community with Economic Dispatch: This scenario will introduce the optimization of resources towards the markets. This enables the comparison of the energy costs of the energy community and impacts in the grid. In addition, there can be a comparison of optimization functions and algorithms.

2) Grid Optimization: The objective is to develop grid control on top of the grid scenario results. This will include real-time congestion management, operational grid optimization and predictive grid optimization functions as a part of Distribution Management System (DMS). Real-time congestion management will make control decisions on how to avoid grid congestion in real-time, i.e., utilize primary controllers that are directly controlled by a distribution grid company. Operational grid optimization has more time to make a decision, because its aim is to enhance the status of acceptable grid status. Its main aim is to coordinate primary controllers owned (e.g., voltage controller of on-load tap changer of a transformer), contracted (e.g., purchased flexibility via local flexibility market from a flexibility service provider) or mandated (e.g., voltage control of a production unit) by a distribution grid company [16]. Furthermore, the plan is to consider near future issues, such as avoiding the operation of on-load tap changer to reduce maintenance cost and considering the activation of purchased conditional flexibility [17]. Predictive grid optimization is utilized to inform flexibility service providers about the flexibility needs (predict day-ahead congestion) and to purchase flexibility from a local flexibility market [18].

3) Grid Optimization with Energy Community and Economic Dispatch: This combines the two earlier mentioned scenarios. The plan is to implement several coordination mechanisms to compare synergies and conflicts between a distribution grid company and energy community.

V. Discussion

As specified in part (1) of the research question (see Section I), the simulation platform (SimCES) enables the comparison of simulation scenarios and the assessment how


```
{ "Type" : "ResourceState",
  "Timestamp" : "2020-10-14T04:04:21.045Z",
  "SimulationId" : "2020-10-14T04:01:52.345Z",
  "SourceProcessId" : "Load3",
  "MessageId" : "Load3_14",
  "EpochNumber" : 14,
  "TriggeringMessageIds" : [ "epoch-14" ],
  "Bus" : "loadbus1",
  "RealPower" :
  { "Value": 100.0, "UnitOfMeasure": "kW" },
  "ReactivePower" :
  { "Value": 0.0, "UnitOfMeasure": "kV.A{r}" } }
```


Fig. 4. Example of a ResourceState message.

the properties of energy processes affect operation and efficiency. Compared to earlier solutions, SimCES takes flexibility and ease of development to a new level. SimCES can operate an arbitrary number of different components that can be parametrized as well as included and excluded from simulation as appropriate. The design follows the SOA principles loose coupling, abstraction, autonomy, reusability, and composability. This reduces the number of dependencies and adds flexibility to the development of simulation scenarios. The simulations can contain functionality from any system level, e.g., Distribution Management System (DMS) can be simulated with consumer-level energy management. The simulation components can be geographically distributed, and the runtime platform can be any. The components run asynchronously and are synchronized as necessary to align the perception of time in simulation.

As simulations are executed, all messages are stored into the Logging System. This enables the analysis of results by providing a Rest API that can generate timeseries from logged data as well as return individual messages. Because everything is in the logs, it is possible to later check which factors have affected the decisions of the algorithms.

Part (2) of the research question, facilitating the development of control algorithms and coordination schemes, is enabled but not yet proven. Although there is currently no proof of concept, the same principles that fulfill part (1) contribute to part (2). On the other hand, as the simulation components execute autonomously, they can even have conflicting goals (part 3). This enables the simulation of how real-life actors operate in electricity networks, as they target at a local optimum. However, such simulations are future work.

The current simulation platform only considers energy systems. However, this could be extended with aspects, such as thermodynamics and communications.

The infrastructure could be in cloud rather than on premises. This would improve scalability especially if the complexity of simulation problems grows.

VI. Conclusions and Future Work

This article presented Simulation Environment of Complex Energy System (SimCES) to distribute the simulation of electricity grid, energy communities, and prosumers to components that communicate via a message bus. SimCES provides a scalable, loosely coupled platform that enables the inclusion and exclusion of energy system actors depending on what is simulated. The message bus applies publish-subscribe communication, which eliminates direct and especially physical dependencies between the components, enabling geographical distribution. Therefore, the system enables real-life, multi-stakeholder co-simulation with industrial algorithms and functions. Besides, it guarantees the consideration of intellectual property rights for simulation participants. The current implementation is functional and can simulate cases that exclude network-wide coordination.

There is still room for future research. More simulation components can be added for more complexity. This will include Economic Dispatch that seeks advantage in energy pricing and flexibility. Furthermore, there will be studies about coordinated control, both vertically within and between stakeholders, by developing processes and rules for flexibility trading in a multi-market, multi-stakeholder environment. The information models could be aligned with existing standards to help in integrating to actual energy systems. Additionally, runtime-platform-independent information models could be developed for coordination and control tasks. Finally, the plan is enable the platform to simulate any domain. Therefore, the user would fetch the platform along with the domain-agnostic components, develop the required domain-specific components required and start simulating.

Acknowledgment

The authors would like to thank the colleagues in ProCemPlus at *Tampere University*, in *VTT Technical Research Centre of Finland* and at *Tampere University of Applied Sciences*, the companies in ProCemPlus steering group and the partners in INTERRFACE H2020 project (http://www.interrface.eu).

References

[1] C. Aura and U. Andreas, "Energy communities: an overview of energy and social innovation," Publications Office of the EU, Tech. Rep., 2020.
[2] "Functional mock-up interface," URL https://fmi-standard.org/ [Retr. 11 Nov 2020].
[3] P. Divshali *et al.*, "Smart grid co-simulation by developing an FMI-compliant interface for PSCAD," in *25th International Conference on Electricity Distribution (CIRED 2019)*, 2019, p. 849.
[4] M. Grigull *et al.*, "A European platform for distributed real time modelling & simulation of emerging electricity systems," Publications Office of the European Union, Tech. Rep., 2016.
[5] A. Kulmala *et al.*, "Information exchange platform for enabling ancillary services from distributed energy resources," in *2019 16th International Conference on the European Energy Market (EEM)*, 2019, pp. 1–5.
[6] L. Zhao, M. Ni, H. Tong, and Y. Li, "Design and application of distributed co-simulation platform for cyber physical power system based on the concepts of software bus and middleware," *IET Cyber-Physical Systems: Theory & Applications*, vol. 5, no. 1, pp. 71–79, 2020.
[7] D. Hästbacka, P. Kannisto, and M. Vilkko, "Data-driven and event-driven integration architecture for plant-wide industrial process monitoring and control," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2018, pp. 2979–2985.
[8] "The mosaik API," URL http://mosaik.readthedocs.io [Retr. 1 Oct 2020].
[9] T. Erl, *Service-oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.
[10] G. Hohpe and B. Woolf, "Enterprise integration patterns - message bus," 2019, URL https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageBus.html [Retr. 8 Jul 2020].
[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, p. 114–131, Jun. 2003.
[12] "RabbitMQ," URL https://www.rabbitmq.com [Retr. 11 Nov 2020].
[13] S. Lu, S. Repo, M. Salmenperä, J. Seppälä, and H. Koivisto, "Using IEC CIM standards and SOA technology for coordinated voltage control application," in *2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE, 2019, pp. 1–5.
[14] "Docker," URL https://www.docker.com [Retr. 12 Nov 2020].
[15] "MongoDB," URL https://www.mongodb.com [Retr. 11 Nov 2020].
[16] S. Repo *et al.*, "The IDE4L project: Defining, designing, and demonstrating the ideal grid for all," *IEEE Power and Energy Magazine*, vol. 15, no. 3, pp. 41–51, 2017.
[17] A. Kulmala *et al.*, "Hierarchical and distributed control concept for distribution network congestion management," *IET Generation, Transmission & Distribution*, vol. 11, no. 3, pp. 665–675, 2017.
[18] M. Attar, A. Supponen, and S. Repo, "Distribution system congestion management through local flexibility market," in *Proceedings of CIRED 2020 Berlin Workshop*, 2020, in press.