Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

SimCES platform for modular simulation: Featuring platform independence, container ecosystem, and development toolkit

Petri Kannisto [*], Ville Heikkilä, Otto Hylli, Mehdi Attar, Sami Repo, Kari Systä

Tampere University, P.O. Box 553, 33014 Tampere University, Finland

ARTICLE INFO

Article history:
Received 13 April 2022
Received in revised form 11 July 2022
Accepted 4 August 2022

Keywords:
Co-simulation
Microservices
Message broker
Docker

ABSTRACT

Modular co-simulation contributes to both engineering and research, but the earlier solutions have lacked the combination of platform independence, loose coupling between the modules, and tools for straightforward development. This paper describes the simulation platform SimCES (Simulation Environment of Complex Energy System) that solves these issues with a microservice architecture, combining message-broker-based communication, containerization, and a development toolkit. The components can even communicate over Internet. Furthermore, there are developer tools that enable an easy start for developers with Python and Docker, but any external platform is possible too. SimCES is domain agnostic but stems from the energy domain.

Code metadata

Current code version	v1.0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00088
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Minimum tested: Docker Engine 20.10.2, Docker Compose 1.28.2; Linux recommended, tested on Ubuntu 20.04.3 LTS (desktop)
If available Link to developer documentation/manual	https://simcesplatform.github.io/
Support email for questions	antti.keski-koukkari@vtt.fi, kari.systa@tuni.fi, petri.kannisto@tuni.fi, sami.repo@tuni.fi

1. Motivation and significance

Simulation systems can receive benefit from a modular, loosely coupled architecture in the spirit of microservices. Simulations are often implemented in a centralized way with a tight coupling between the components simulating the related sub-problems. Tight coupling refers to features, such as point-to-point dependencies, synchronous computation, centralized control, static binding, strong platform dependencies, and the simultaneous deployment of components [1, Ch. 4]. Such systems serve inadequately when the components

- represent several independent, competing optimizers,
- are developed by separate teams that are experts in their own domain,
- are developed and deployed independently from each other,
- are best executed in different hardware and software platforms, possibly in separate locations.

These requirements could be met with a system inspired from microservice architectures [2].

This paper introduces Simulation Environment of Complex Energy System (SimCES), an open source platform that facilitates the development of complex simulation systems with a modular, message-broker-based architecture. SimCES provides loose coupling between the components, enabling the developers to focus on their component only, because the architecture takes care of component parametrization and synchronization during execution. Modular simulation approaches are often referred to as *co-simulation* [3]. However, often co-simulation refers to tight coupling with no distribution over a network. Instead, SimCES enables the simulation components to execute on any platform as long as they implement the required workflow and

[*] Corresponding author.
E-mail address: petri.kannisto@tuni.fi (Petri Kannisto).

API, built upon a message broker. Furthermore, platform independence adds flexibility to environments, enabling external simulation systems and even server clusters to be connected for resource-intensive calculation. The simulations are configured with a human-readable file that defines the parameters and included components, facilitating the comparison of simulation results in varying scenarios. The other advantages include a container ecosystem for reusable simulation images as well as a programming toolkit to reduce the development effort. The objectives of SimCES can be summarized as follows:

> What kind of software platform enables the development of complex simulation systems when the requirements are modularity, loose coupling, platform independence, distribution over Internet, configurability, and parametrization?

SimCES is not restricted to any particular application area although energy domain was the initial motivator. Earlier, the core idea of SimCES was explained in [4], but this was work in progress, still lacking the container ecosystem concept, the development toolkit, and the current simulation cases.

Next, Section 2 reviews related research. Then, Section 3 explains the design of SimCES. Section 4 introduces concrete applications, followed by a discussion about the impact in Section 5. Finally, Section 6 concludes the article.

2. Related research

The earlier simulation platforms lack the combination of loose coupling, platform independence, domain-agnostic design, and the possibility to build a domain ecosystem from simulation components. Mosaik API enables distributed simulation but connects the components via sockets [5]. These connect point-to-point and therefore lack scalability in large-scale systems. iTETRIS was developed to enable distributed simulation for intelligent transportation systems [6]. While distributed and configurable, it was designed for a single domain. SimApi has been developed exclusively for the energy management of buildings, and its architecture lacks the single-protocol-for-all-communication approach of SimCES [7]. DEMKit was developed exclusively for energy management and is a toolkit rather than a platform for distributed simulation [8]. MOOSE is a platform for modular, parallel multiphysics simulations but integrates the modules into one application over C++ APIs [9]. The co-simulation system LICPIE builds upon a middleware with similarities to SimCES but lacks the publish–subscribe approach and aims at connecting simulators rather than proving a platform for simulation components of any size [10]. Despite modularity, SpaceCRAFT VR is centralized and communicates over sockets, therefore lacking any actual protocol support to build scalable systems over Internet [11]. COCOP showed the applicability of a message broker in distributed simulation but is an industrial integration framework rather than a simulation platform [12]. Spine Toolbox is a simulation and modeling tool rather than a platform [13]. As far as is known, SimCES is the only platform enabling a component-based domain ecosystem. Of the reviewed simulation software, all but LICPIE and SpaceCRAFT VR are open source similar to SimCES.

Even open standards exist for co-simulation. Functional Mock-up Interface (FMI) is a commonly used interface specification [14]. Compared to SimCES, FMI lacks an explicit support for network communication, which would facilitate interoperability and distribution. Still, FMI can be complemented with "FMIGo!" that provides a backend and enables networking [15]. Compared to the message-broker-based SimCES, "FMIGo!" builds the network communication upon ZeroMQ [16], which is brokerless and therefore provides no Internet-wide message routing. Distributed Co-simulation Protocol (DCP) is another technology for networked distribution [17]. However, the included protocols lack message routing, and DCP assumes central orchestration whereas SimCES is distributed.

The agent-based approach provides an alternative to simulate interaction. The concept *agent-based model* refers to modeling the behavior of intelligent actors [18]. Respectively, *agent-based modeling and simulation* emphasizes the utilization of agents for simulation purposes [19]. These approaches build upon an interface scheme made specifically for agents, whereas SimCES uses interfaces similar to the respective real-life software systems, enabling simulation or emulation in software-in-the-loop fashion.

SimCES builds upon a microservice architecture but can be considered a special case. The microservice architecture is a distributed design approach where the software components are fine grained, communicate over simple channels, as well as are executed and deployed independent of each other [20]. Microservices bring scalability as well as agility to development and deployment, but more of governance is required compared to a monolith [21]. There are dozens of design patterns for microservices, of which at least asynchronous messaging, service discovery (with topics in the message broker), externalized configuration, and log aggregator have been applied in SimCES [22]. Microservices can communicate over multiple patterns, of which publish–subscribe, message-oriented middleware, and asynchronous communication apply to SimCES in contrast to the more typical synchronous Restful approach [23]. Eventually, SimCES is a special case of microservices due to not only its differences in deployment and instantiation but also the lifecycle of execution. SimCES was not designed to host services with a long uptime but it rather assumes a re-instantiation for each simulation run. Depending on what is simulated, the simulation run typically takes from a few seconds to a few hours, and the user presumably monitors the execution. This condition is a relaxation compared to any continuously operating microservice architecture that should consider, e.g., resilience, fault tolerance, and deployment at runtime. That is, SimCES can be considered a manually executed microservice system.

3. Software description

This section explains the architecture of SimCES, first presenting the fundamental principles and technologies. This is followed by software platforms, component management, and the development toolkit.

3.1. Architecture fundamentals

Any co-simulation necessitates information exchange between the components, and SimCES implements this with a message broker using the topic-based publish–subscribe pattern. The broker enables loose coupling, effectively hiding the components from each other and removing point-to-point connections [24]. That is, to communicate, each component only needs the topics to publish to and to listen to. The actual message routing occurs in the broker, reducing the communication burden of the components.

Because the message broker provides loose coupling in time, special attention is necessary to synchronize the components. For example, when the platform simulates energy consumption during a particular hour in a day, the components must be made aware of the time. This is reached with *epochs*, periods of simulated time that the platform communicates to the components. An epoch starts when the platform instructs this and ends when all components have finished the related calculation. The length of epochs can be set based on the needs of the simulation. The epoch concept is explained in detail in [4].

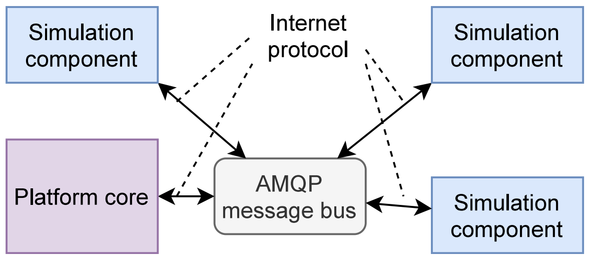

Fig. 1. The AMQP message broker enables the simulation components to be distributed over Internet.

Technically, the messaging is based on Advanced Message Queueing Protocol (AMQP), and the messages are encoded in JavaScript Object Notation (JSON). AMQP 0-9-1 is an Internet standard (supported by, e.g., RabbitMQ [25]), whereas JSON is among the most popular serialization formats. This standard-based design enables any software platform for the components as well as any geographical location (see Fig. 1).

The platform includes a logging system that records all messages sent during simulation. After a simulation run, the user can retrieve these messages to analyze the results. The logging system can even generate timeseries from similarly structured messages published to the same topic, facilitating the analysis of the results. For example, to follow how the energy consumption of a simulated load has changed, the user would generate a timeseries from the consumption messages published from the component. The timeseries can be generated in either JSON or Comma Separated Values (CSV) for a spreadsheet.

3.2. Platforms and image registries

SimCES enables the components to execute either in the platform as Docker containers (*platform managed*) or any other environment (*externally managed*). If needed, a single simulation can apply both approaches. Containerization facilitates the setup of components, but if a component requires a specific platform, it must be externally managed. The containerized components can be either built locally or retrieved from a public image registry. That is, there are three types of component origins: externally managed platform, public or private image registry, and local build (see Fig. 2). The core of the platform itself, which manages component execution and synchronization, is deployed as Docker containers.

To enable the platform to operate an arbitrary component developed by anyone, the component must provide a manifest for an interface. This includes the name, type, and description of the component as well as Docker image name or location for platform-managed components. Additionally, the manifest can define attributes to be delivered at startup to enable the parametrization of simulations. For platform-managed components, these are injected as environment variables, whereas externally managed components receive these in a dedicated message from the broker. The format of the manifest is YAML (recursive acronym from *YAML Ain't Markup Language*), which enables editing even by non-technical users. An example manifest is available at [26].

3.3. Development Toolkit

The implementation of all the required interactive workflows necessitates effort. The workflows must cover component parametrization, the start and end of execution, and component synchronization.

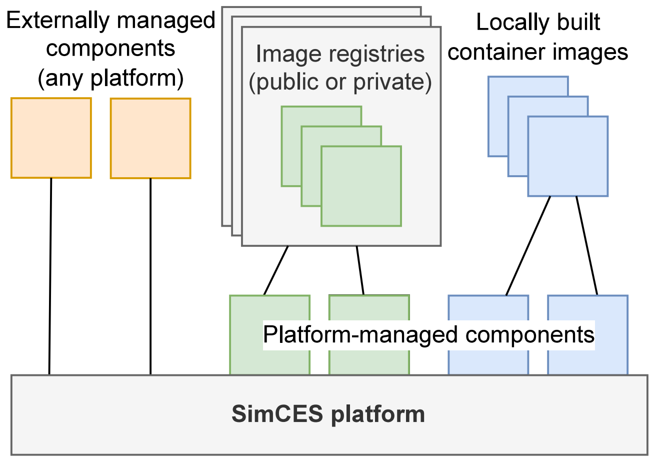
Externally managed components (any platform)
Image registries (public or private)
Locally built container images
Platform-managed components
SimCES platform

Fig. 2. The components can be externally managed systems or platform-managed containers, the container images either retrieved from a container registry or built locally.

For faster implementation, the developers can take advantage from SimCES toolkit that implements the basic requirements of communication. Currently, the toolkit has a Python implementation with the following main features:

- Proxy classes for message structures, covering:
  - The messages needed for communication with the platform (e.g., epochs and simulation start and end)
  - The common metadata fields that appear frequently in messages (e.g., timestamp, the ID of source component, and epoch number)
- A client class for the message broker, creating an abstraction layer to encapsulate the protocol-specific complexity
- An abstract base class for the basic workflow of components; the component developer extends this with component-specific functionality

While available for Python only, the toolkit can be developed for additional languages as needed. In SimCES documentation [27], the Python implementation is called *Simulation Tools*.

3.4. Iterative simulation

The specification of SimCES supports even iterative calculation among multiple components. In this scheme, it is agreed at design time that two or more components accept intermediate results from each other via appropriate topics. An iterating component can be either active or passive, which means that an active component can decide when to end iteration, whereas passive components simply iterate until an active component decides to end. The end is indicated in a dedicated field in the message that contains the calculated result. If the iteration fails to converge, the provided final result may be calculated with a backup method. To facilitate message filtering, the topic structure enables subscribers to receive intermediate results, final results, or both.

For more information about iteration, see [27].[1] However, there are currently no prototypes to apply the specified iteration scheme.

[1] Exact link to iteration specification: https://simcesplatform.github.io/core_workflow-sim/

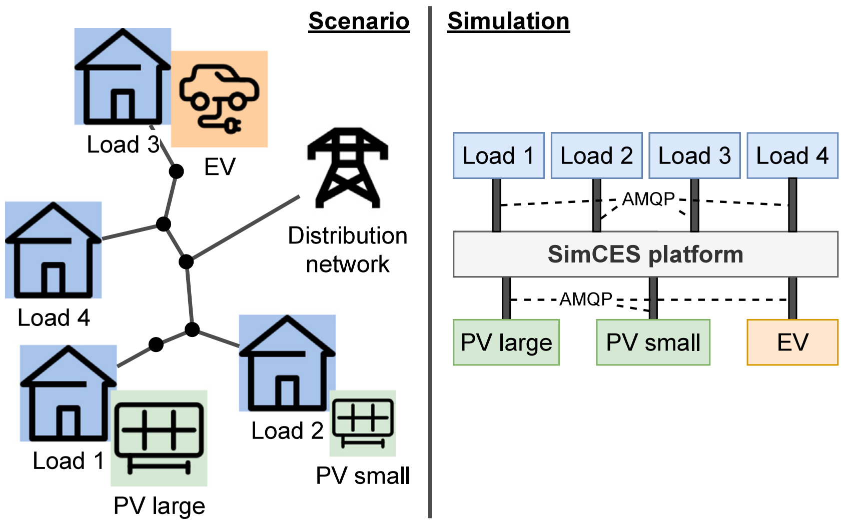

Fig. 3. The energy community (left) and the respective simulation components (right).

4. Applications

This section explains how SimCES operates in actual applications. While only two cases are included, there are more being developed.

4.1. Platform fundamentals with energy community

Although SimCES was originally packaged as separate components, this article is bundled with the "standalone" version that ships in one repository. Still, this standalone version is functionally identical to the original one.

The standalone version includes the code required to build an energy community demonstration for experiments. The term *energy community* has multiple definitions and can be divided into various categories [28]. Generally, the energy community refers to a group of energy consumers or producers (or prosumers) that operate together to generate shared value [29]. However, the demonstration solely includes the communication of data from predefined timeseries rather than actual simulations, as the main purpose is to show how the platform operates.

The scenario presents an imaginary energy community of four detached houses connected to the power grid as illustrated on the left side of Fig. 3. Respectively, the right side how the functionality maps to simulation components. The four houses consume electricity. In addition, two of them supply solar power with photovoltaic equipment (PV "large" and "small") whereas one charges an electric vehicle (EV). The scenario is explained in more detail in the documentation [27]. To execute the scenario, refer to the readme file shipped with the standalone version. Not only the scenario but also SimCES platform were developed in the project ProCemPlus [30].

4.2. More complex simulations

Another use case involves a study for congestion management in electricity distribution, referring to the prevention of situations where the quality of electricity would deviate. A congestion could, for example, lead to a non-acceptable voltage or current level. Here, congestion management occurs with a market mechanism, that is, the operator of the distribution system announces flexibility needs to avoid anticipated congestion whereas energy communities make related bids [31]. The included simulation components are predictive grid optimization, local flexibility market, grid simulator, energy communities, and state monitoring,

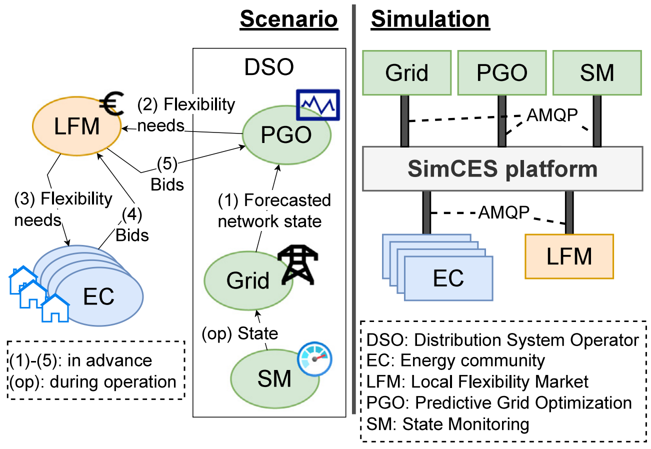

Fig. 4. The congestion management scenario (left) and the respective simulation components (right).

some of which are platform managed and the others externally managed. Table 1 shows component details, whereas Fig. 4 illustrates communication. The study is funded by European Union (EU) via the project INTERRFACE [32].

A few scenarios have been simulated with some early results, and the experience of SimCES is as follows. The under-study distribution grid, in one scenario, was an actual electric power distribution grid located in a small town in Finland. The duration of the simulation was one week with a one-hour resolution. The execution took approximately 45 min leading to 915,000 published messages. SimCES operated without any lag, and the messages were accessible afterwards through the logging system. Concerning the development process, the access to Docker logs facilitated the development and testing of platform-managed components. Due to the multidisciplinarity of the simulation, there was a need for the cooperation of several developers with different expertise, which was facilitated by SimCES thanks to its distributed architecture. Nevertheless, the communication between the developers was challenging when aiming to reach a common understanding of the functionalities and interactions between components. In addition, maintaining comprehensive and up-to-date documentation proved essential in saving effort in the development.

Table 1
The components in the congestion management simulation.

Component	Functionality	Implementation	Managed
Energy community	Offers flexibility	Python	By platform
Grid	Represents and replicates the physical grid	OpenDSS simulator with Python wrapper	Externally
Local flexibility market	Connects flexibility buyers and sellers	Python	By platform
Predictive grid optimization	Detects flexibility needs in advance and selects bids from the market	Matlab	Externally
State monitoring	Monitors the grid in real time	Matlab	Externally

Future research will cover more of cases. This includes at least value sharing mechanisms in an energy community.

5. Impact

SimCES is a novel approach for distributed simulations, based on a powerful paradigm. The advantages of the message broker, particularly loose coupling, platform independence, and possibility for geographical distribution, provide an advantageous combination in complex scenarios with heterogeneous software platforms.

Containerization introduces a possibility to develop re-usable simulation components to form an entire ecosystem [33]. For instance, a simulation component can implement a model for energy storages. This would be shared via an image registry, such as Docker Hub or Github container registry, and deployed into simulations as needed. Such a storage component was already developed for SimCES [26]. In general, containerization is considered a tool for modular software development, helping in management [34]. Due to containerization, SimCES brings the infrastructure as code (IaC [35]) aspect into simulations, enabling the automatic setup of the infrastructure and therefore facilitating deployments. Although this advantage lacks from externally managed components, presumably most components are developed as container images and any complex calculation, requiring another platform, is merely a subset among the components.

The microservice approach carries both advantages and disadvantages in co-simulation. While there is agility in development, governance is necessary in development, and integration testing can be challenging [21]. Different languages and technologies are possible, the components are physically isolated, and the application size is unlimited (especially in externally managed components), but the communication interfaces require implementation effort and cause latency, the users should be authenticated and authorized at runtime, and the whole is more complex compared to non-Internet interfaces or a monolith [36]. These points apply to not only microservice architectures in general but also SimCES.

Due to microservices, SimCES is inevitably more difficult to domain experts compared to simulators that operate over straightforward application-to-application interfaces, but SimCES still provides supportive tools. Any developer accustomed to existing widespread tools, such as FMI, Modelica, or Matlab, needs a paradigm shift to adopt SimCES. However, while easier, these tools or any platform reviewed in Section 2 lack the advantages of SimCES. Additionally, even these simulators can be connected to SimCES if needed to become a part of a multi-simulator scheme. To help domain experts in development, there can be an ICT professional to help in creating the required connections (which is likely no different from typical teams developing complex simulators). On the other hand, SimCES offers a toolkit for Python to help the developers, which is a clear benefit because Python provides tools for scientific calculation. Python is the most popular language as of July 2022 [37]. For Matlab, a widely adopted calculation and simulation tool, there is already a SimCES connector called *AmqpMathToolIntegration* [27].

As AMQP is an Internet protocol, there is no mechanism to guarantee the time consumed for each message delivery, which determines the real-time characteristics of SimCES. Internet protocols can neither reserve any bandwidth nor avoid congestion in the network if such occurs. For more of determinism, the message broker could operate in a closed network with some capacity reserve and no competing traffic. While this may enable millisecond-scale delivery times with powerful hardware, there is no intention for hard real-time systems due to the lacking time guarantees. If speed were the goal, a brokerless protocol would eliminate the overhead from message routing, but this would as well remove the characteristic of loosely coupled publish–subscribe communication along with the topic-based discovery.

6. Conclusions

SimCES is a simulation environment that enables a domain ecosystem of microservice-style simulation components, enabling the components to be distributed as container images. Furthermore, SimCES enables simulation systems to span across software platforms and locations, providing the tools required to manage the system as a whole. Any software platform is possible, and the development toolkit (currently implemented in Python) provides a low-effort way for the development of simulation components. The microservice architecture has both advantages and disadvantages, such as platform independence versus increased complexity in communication, possibility of distributed development versus requirement of developer coordination, and distribution over network versus no support for hard real time. The capabilities of the platform have been evaluated with case studies in the energy domain.

In near future, SimCES will provide further results in ongoing research projects in the energy domain. This includes at least congestion management in future energy systems and energy management in energy communities.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data required to experiment with the system is shared along with the linked source code.

Acknowledgments

This paper was financially supported by the national project ’Prosumer Centric Energy Communities towards Energy Ecosystem’ (ProCemPlus) funded by Business Finland (diary 842/31/2019), the European project H2020 ‘INTERRFACE’ (TSO-DSO-Consumer INTERFACE aRchitecture to provide innovative grid services for an efficient power system, grant no. 824330), and the national project ’Distributed Management of Electricity System’ (DisMa) funded by Academy of Finland (grants no. 322673 and no. 322676). The authors want to thank the funders, including the companies in ProCemPlus steering group. Furthermore, the authors thank the project partners at Tampere University and VTT Technical Research Centre of Finland, especially (in alphabetical order): Timo Aaltonen, Antti Keski-Koukkari, Anna Kulmala, Ville Mörsky, Amir Safdarian, Olli Suominen, Antti Supponen, Md Tanjimuddin, and Matti Vilkko.

References

[1] Josuttis NM. SOA in practice. O'Reilly Media, Inc. 2007.

[2] Taibi D, Lenarduzzi V, Pahl C. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. IEEE Cloud Comput 2017;4(5):22–32. http://dx.doi.org/10.1109/MCC.2017.4250931.

[3] Gomes C, Thule C, Broman D, Larsen PG, Vangheluwe H. Co-simulation: A survey. ACM Comput Surv 2018;51(3). http://dx.doi.org/10.1145/3179993.

[4] Kannisto P, Hylli O, Heikkilä V, Supponen A, Aaltonen T, Repo S, et al. Software and communications platform for simulation environment of complex energy system (SimCES). In: 2021 IEEE madrid powertech. 2021, p. 1–6. http://dx.doi.org/10.1109/PowerTech46648.2021.9495020.

[5] Lehnhoff S, Nannen O, Rohjans S, Schlogl F, Dalhues S, Robitzky L, et al. Exchangeability of power flow simulators in smart grid co-simulations with mosaik. In: 2015 Workshop on modeling and simulation of cyber-physical energy systems. 2015, p. 1–6. http://dx.doi.org/10.1109/MSCPES.2015.7115410.

[6] Rondinone M, Maneros J, Krajzewicz D, Bauza R, Cataldi P, Hrizi F, et al. iTETRIS: A modular simulation platform for the large scale evaluation of cooperative ITS applications. Simul Model Pract Theory 2013;34:99–125. http://dx.doi.org/10.1016/j.simpat.2013.01.007.

[7] Pallonetto F, Mangina E, Milano F, Finn DP. SimApi, a smartgrid co-simulation software platform for benchmarking building control algorithms. SoftwareX 2019;9:271–81. http://dx.doi.org/10.1016/j.softx.2019.03.003.

[8] Hoogsteen G, Hurink JL, Smit GJM. DEMKit: a decentralized energy management simulation and demonstration toolkit. In: 2019 IEEE PES innovative smart grid technologies Europe. 2019, p. 1–5. http://dx.doi.org/10.1109/ISGTEurope.2019.8905439.

[9] Permann CJ, Gaston DR, Andrš D, Carlsen RW, Kong F, Lindsay AD, et al. MOOSE: Enabling massively parallel multiphysics simulation. SoftwareX 2020;11:100430. http://dx.doi.org/10.1016/j.softx.2020.100430.

[10] Zhao L, Ni M, Tong H, Li Y. Design and application of distributed co-simulation platform for cyber physical power system based on the concepts of software bus and middleware. IET Cyber-Phys Syst Theory Appl 2020;5(1):71–9. http://dx.doi.org/10.1049/iet-cps.2018.5084.

[11] Jakubik CR, Johnston A, Zhong P, McHenry N, Chamitoff G. SpaceCRAFT VR: an event-driven, modular simulation platform with fully-asynchronous physics. In: AIAA scitech 2021 forum. 2021, p. 1–16. http://dx.doi.org/10.2514/6.2021-0898.

[12] Kannisto P, Hästbacka D, Gutiérrez T, Suominen O, Vilkko M, Craamer P. Plant-wide interoperability and decoupled, data-driven process control with message bus communication. J Ind Inf Integr 2022;26:100253. http://dx.doi.org/10.1016/j.jii.2021.100253.

[13] Kiviluoma J, Pallonetto F, Marin M, Savolainen PT, Soininen A, Vennström P, et al. Spine toolbox: A flexible open-source workflow management system with scenario and data management. SoftwareX 2022;17:100967. http://dx.doi.org/10.1016/j.softx.2021.100967.

[14] Junghanns A, Gomes C, Schulze C, Schuch K, Mai PR, Blaesken M, et al. The functional mock-up interface 3.0 - new features enabling new applications. In: Proceedings of 14th Modelica conference 2021. 2021, p. 17–26. http://dx.doi.org/10.3384/ecp2118117.

[15] Lacoursière C. FMI go! A simulation runtime environment with a client server architecture over multiple protocols. In: Proceedings of the 12th international modelica conference. 2017, p. 653–62. http://dx.doi.org/10.3384/ecp17132653.

[16] ZeroMQ, URL http://zguide.zeromq.org. [Retrieved 7 Jul 2022].

[17] Krammer M, Benedikt M, Blochwitz T, Alekeish K, Amringer N, Kater C, et al. The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In: Proceedings of the 50th computer simulation conference. SummerSim '18, Society for Computer Simulation International; 2018, p. 1–14.

[18] Ferreira dos Santos A, Saraiva J. Agent based models in power systems – a literature review. U Porto J Eng 2020;7(3):101–13. http://dx.doi.org/10.24840/2183-6493_007.003_0009.

[19] Abar S, Theodoropoulos GK, Lemarinier P, O'Hare GM. Agent based modelling and simulation tools: A review of the state-of-art software. Comput Sci Rev 2017;24:13–33. http://dx.doi.org/10.1016/j.cosrev.2017.03.001.

[20] Lewis J, Fowler M. Microservices. 2014, URL https://martinfowler.com/articles/microservices.html. [Retrieved 11 Jul 2022].

[21] Baškarada S, Nguyen V, Koronios A. Architecting microservices: Practical opportunities and challenges. J Comput Inf Syst 2020;60(5):428–36. http://dx.doi.org/10.1080/08874417.2018.1520056.

[22] Valdivia JA, Lora-González A, Limón X, Cortes-Verdin K, Ocharán-Hernández J. Patterns related to microservice architecture: a multivocal literature review. Program Comput Soft 2020;46:594–608. http://dx.doi.org/10.1134/S0361768820080253.

[23] Karabey Aksakalli I, Celik T, Can AB, Tekinerdogan B. Deployment and communication patterns in microservice architectures: A systematic literature review. J Syst Softw 2021;180:111014. http://dx.doi.org/10.1016/j.jss.2021.111014.

[24] Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M. The many faces of publish/subscribe. ACM Comput Surv 2003;35(2):114–31. http://dx.doi.org/10.1145/857076.857078.

[25] RabbitMQ. 2022, URL https://www.rabbitmq.com/. [Retrieved 31 Jan 2022].

[26] Storage resource. 2021, URL https://github.com/simcesplatform/storage-resource. [Retrieved 3 Feb 2022].

[27] Simulation environment of complex energy system (SimCES). 2022, URL https://simcesplatform.github.io/. [Retrieved 7 Jul 2022].

[28] Valta J, Kulmala A, Järventausta P, Kirjavainen J, Mäkinen S, Björkqvist T, et al. Towards practical typology of energy communities: main differentiating elements and examples of promising implementations. In: CIRED 2021 - the 26th international conference and exhibition on electricity distribution. United Kingdom: Institution of Engineering and Technology; 2021, p. 3196–200. http://dx.doi.org/10.1049/icp.2021.1712.

[29] Kulmala A, Baranauskas M, Safdarian A, Valta J, Järventausta P, Björkqvist T. Comparing value sharing methods for different types of energy communities. In: 2021 IEEE PES innovative smart grid technologies Europe. 2021, p. 1–6. http://dx.doi.org/10.1109/ISGTEurope52324.2021.9640205.

[30] ProCemPlus project. 2022, URL http://www.senecc.fi/projects/procemplus. [Retrieved 17 Mar 2022].

[31] Attar M, Repo S, Mann P. Congestion management market design- approach for the nordics and central Europe. Appl Energy 2022;313:118905. http://dx.doi.org/10.1016/j.apenergy.2022.118905.

[32] INTERRFACE project. 2022, URL http://www.interrface.eu/. [Retrieved 7 Mar 2022].

[33] Syed MH, Fernandez EB. A reference architecture for the container ecosystem. In: Proceedings of the 13th international conference on availability, reliability and security. ARES 2018, New York, NY, USA: Association for Computing Machinery; 2018, p. 1–6. http://dx.doi.org/10.1145/3230833.3232854.

[34] Koskinen M, Mikkonen T, Abrahamsson P. Containers in software development: A systematic mapping study. In: Franch X, Männistö T, Martínez-Fernández S, editors. International conference on product-focused software process improvement. Cham: Springer International Publishing; 2019, p. 176–91.

[35] Rahman A, Mahdavi-Hezaveh R, Williams L. A systematic mapping study of infrastructure as code research. Inf Softw Technol 2019;108:65–77. http://dx.doi.org/10.1016/j.infsof.2018.12.004.

[36] Taibi D, Lenarduzzi V, Pahl C. Architectural patterns for microservices: A systematic mapping study. In: Proceedings of the 8th international conference on cloud computing and services science - CLOSER. SciTePress, INSTICC; 2018, p. 221–32. http://dx.doi.org/10.5220/0006798302210232.

[37] TIOBE index for July 2022. 2022, URL https://www.tiobe.com/tiobe-index/. [Retrieved 11 Jul 2022].