A comparison study of co-simulation frameworks for multi-energy systems: the scalability problem

Barbierato *et al. Energy Informatics* 2022, **5**(Suppl 4):53
https://doi.org/10.1186/s42162-022-00231-6

Energy Informatics

REVIEW

Open Access

A comparison study of co-simulation frameworks for multi-energy systems: the scalability problem

Luca Barbierato[1*], Pietro Rando Mazzarino[2], Marco Montarolo[2], Alberto Macii[2], Edoardo Patti[2] and Lorenzo Bottaccioli[1]

From Energy Informatics.Academy Conference 2022 (EI.A 2022)
Vejle, Denmark. 24-25 August 2022

*Correspondence:
luca.barbierato@polito.it

[1] Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
[2] Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy

Abstract

The transition to a low-carbon society will completely change the structure of energy systems from a standalone hierarchical centralised vision to cooperative and distributed Multi-Energy Systems. The analysis of these complex systems requires the collaboration of researchers from different disciplines in the energy, ICT, social, economic, and political sectors. Combining such disparate disciplines into a single tool for modeling and analyzing such a complex environment as a Multi-Energy System requires tremendous effort. Researchers have overcome this effort by using co-simulation techniques that give the possibility of integrating existing domain-specific simulators in a single environment. Co-simulation frameworks, such as Mosaik and HELICS, have been developed to ease such integration. In this context, an additional challenge is the different temporal and spatial scales that are involved in the real world and that must be addressed during co-simulation. In particular, the huge number of heterogeneous actors populating the system makes it difficult to represent the system as a whole. In this paper, we propose a comparison of the scalability performance of two major co-simulation frameworks (i.e. HELICS and Mosaik) and a particular implementation of a well-known multi-agent systems library (i.e. AIOMAS). After describing a generic co-simulation framework infrastructure and its related challenges in managing a distributed co-simulation environment, the three selected frameworks are introduced and compared with each other to highlight their principal structure. Then, the scalability problem of co-simulation frameworks is introduced presenting four benchmark configurations to test their ability to scale in terms of a number of running instances. To carry out this comparison, a simplified multi-model energy scenario was used as a common testing environment. This work helps to understand which of the three frameworks and four configurations to select depending on the scenario to analyse. Experimental results show that a Multi-processing configuration of HELICS reaches the best performance in terms of KPIs defined to assess the scalability among the co-simulation frameworks.

Keywords: Co-simulation framework, Scalability, Mosaik, HELICS, AIOMAS

Introduction

According to the United Nations Habitat, cities consume about 78% of global energy demand and generate more than 60% of greenhouse gas emissions primarily through the consumption of fossil fuels for energy supply and transportation (United Nations 2022). To reach the ambitious goals of the Glasgow agreement (Authors 2021), a drastic reduction in carbon emission is needed. To achieve such a reduction, a transition from classic fossil fuels to Renewable Energy Sources (RES) as well as the adoption of integrated energy system components, such as micro co-generators, are required. This transition will completely change the structure of the energy systems from standalone hierarchical centralised energy systems to cooperative and distributed energy systems, the so-called Multi-Energy System (MES) vision. Such a transition can not be left to chance and the development of novel Information and Communication Technology (ICT) tools, platforms, and frameworks for driving this transition are attracting a strong research effort from the scientific community. In the last decades, researchers have given a great effort in the development of domain-specific simulation tools designed to simulate with high efficiency and accuracy the behavior of a particular energy system aspect (Ringkjøb et al. 2018). In MES context, the simulation of different energy systems will require a broader vision and, consequently, a larger number of domains from different systems involved. According to "Smart Grid Architectural Model" (SGAM) (Bruinenberg et al. 2012) and in particular to its extension "General-puprose Architectural Model for Multi Energy Systems" GAMES (Barbierato et al. 2020) in Fig. 1, domains represents the overall conversion chain of an energy carrier that are: (i) generation, (ii) transmission, (iii) distribution, (iv) Distributed Energy Resources (DER), and (v) customer premises. Moreover, the analysis of complex MES requires the collaboration of researchers from

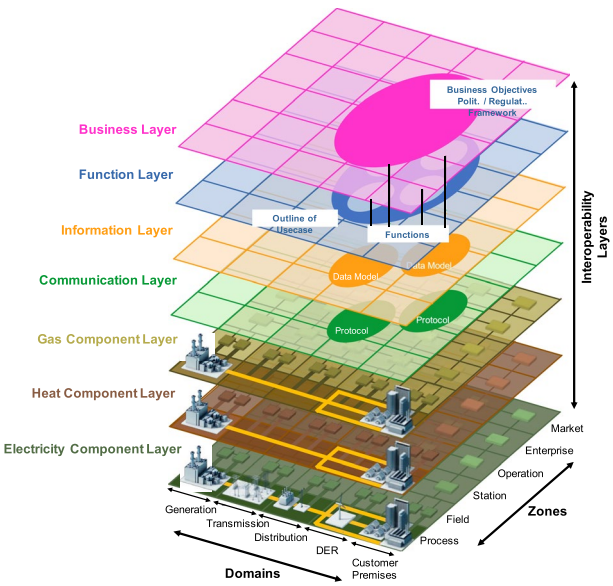

Fig. 1 General-purpose architectural model for MES engineering application (GAMES)

different disciplines applying different perspectives in the energy, ICT, social, economic, and political sectors. Therefore, researchers exploit co-simulation frameworks that must address (i) different domains for each individual energy system, (ii) different energy systems together (e.g. power grid, district heating, gas grid), and (iii) different perspectives of the overall MES (e.g. ICT, energy, economic and social) (Schloegl et al. 2015)

Many works have focused on the co-simulation of smart grids by integrating simulators of power grids with ICT communication aspects, so-called Cyber-Physical Energy System (CPES) (Georg et al. 2013; Garau et al. 2018; Pan et al. 2016; Barbierato et al. 2020). Co-simulation has been widely applied also to integrate several models in order to represent and describe the planning of new RES deployment (Reinbold et al. 2019; Steinbrink et al. 2019; Bottaccioli et al. 2017; Schiera et al. 2019) or to study the effects of novel control strategies to exploit energy flexibility for demand response applications (Song et al. 2017; Bhattarai et al. 2016; Abgottspon et al. 2018; Mazzarino et al. 2021). To ease the coupling of simulators, researchers have started defining standards for co-simulation, such as Functional Mock-up Interface (FMI) (Blochwitz et al. 2011), and co-simulation frameworks, such as Mosaik (Schütte et al. 2011) and HELICS (Palmintier et al. 2017). In particular, Mosaik and HELICS are gaining much attention from the energy research community and were used by several research projects focused on MES. The coordination of domain-specific simulators through co-simulation frameworks can help the development of digital twin platforms for MES (Palensky et al. 2021) that can be used to plan and operate this transition. However, such platforms will require the ability of domain-specific simulation models and co-simulation frameworks to scale up as much as possible to best represent the complexity and interdependencies of very large real systems. For instance, scalability is essential when testing the impact on the power grid of an innovative heat pump technology on a realistic scenario of one million buildings with photovoltaic installations on the rooftops. The present work is indented to compare the effectiveness of the Mosaik and HELICS co-simulation frameworks and the AIOMAS Multi-Agent System (MAS) library (Scherfke 2014) in scaling up the number of entities in a co-simulation environment, evaluating different possible configurations of their usage for parallelizing a simple Python simulator. The choice of these technologies depends mainly on their dominant role in energy sectors among other solutions and their ease of use. In fact, Mosaik and HELICS are popular co-simulation frameworks in the literature for Smart grids (Mihal et al. 2022) and unlike the other solutions they are also thought to be extended to MES or general purpose applications [e.g. (Widl et al. 2022; Sergi and Pambour 2022)]. To the best of our knowledge, this is the first work that tries to benchmark these two co-simulation frameworks (i.e. HELICS and Mosaik) with respect to their scalability performance.

Moreover, this study includes a particular implementation of AIOMAS as a viable alternative to build a co-simulation framework. The choice to include AIOMAS in this study follows the recent trends of coupling co-simulation and MAS concepts (Jung et al. 2018; Paris et al. 2017; Motie et al. 2018; Camus et al. 2016). In particular, from our findings, AIOMAS is the only well-documented and easy-to-use Python library that enables the deployment of MAS with powerful capabilities regarding agent distribution and communication infrastructure. AIOMAS incorporates different abstraction layers that ensure a proper Time Regulation, Synchronization, and Data Exchange Management of

the MAS setup. These additional layers are very powerful and allow the proposed parallelism between MAS and Co-simulation Framework application. Steinbrink et al. (2018) present a comparison between Mosaik and implementation of the IEEE 1516 High-Level Architecture (HLA), which is similar to HELICS implementation. In particular, the objective was to provide researchers with guidelines to assess which of the two implementation suits their needs. They compared both the framework architectural concepts and the accuracy results from a Smart Grid co-simulation study over a representative power system scenario. The authors conclude that implementing benchmarks and deriving a comparative performance analysis of the co-simulation frameworks is worth investigating for future works. In fact, our study fulfills this gap focusing in particular on the scalability aspect of the three above-mentioned frameworks.

The rest of the paper is organised as follows: "Enabling technologies for co-simulation environments" section presents Mosaik and HELICS co-simulation frameworks and the AIOMAS implementation; "Methodology for benchmarking design" section better discusses the problem of scalability, presenting the different co-simulation framework configurations, and the bench-marking metrics; "Setup of co-simulation scenario" section presents the simulators involved in the MES scenario to the purpose of the scalability benchmarking; "Experimental results" section instead presents the experimental results of the benchmark and a qualitative comparison of the analysed frameworks in implementing a co-simulation scenario; finally, "Conclusion" section provides our concluding remarks.

Enabling technologies for co-simulation environments

The co-simulation approach is effective when dealing with multi-domain complex systems in which analytical assessment is no longer feasible considering their complexity. Co-simulation is often related to Cyber-Physical Systems (CPS) (Palensky et al. 2017) and, in particular, Cyber-Physical Energy Systems (CPES) (Zhang et al. 2020), of which the most prominent example can be found in the Smart Grid concept. General notions about co-simulation are thoroughly reported in Gomes et al. (2018) and Schweiger et al. (2019). In these literature definitions, co-simulation allows integrating together heterogeneous domain-specific Simulators creating a shared simulation environment. Therefore, this paradigm allows decomposing a complex system in a System-of-System (SoS) structure by applying system engineering. Each of the identified sub-systems deals with a well-defined problem while interacting with each other. From this perspective rises our parallelism with MAS. In literature, some integration of MAS simulators in Co-simulation frameworks can be found, but by abstracting a little more the concept of MAS it is possible to see a co-simulation framework as a system in which really complex and different agents (e.g. the simulators) interact among each other. Agents are thought of as software components that perform computations and virtually mimic the actions and interactions of real-world systems. Usually MAS agents are considered intelligent components, but abstracting from this definition the main characteristics are autonomy, responsiveness and proactivity (Coelho et al. 2017). These characteristics, despite the level of human-like intelligence, could be applied to the subsystems operating in complex macro-system environments, such as Smart Grids or MESs.

Besides the different co-simulation frameworks such as Mosaik and HELICS, which have different functions and implementations, a shared general architecture can be highlighted.

The main components required to build a co-simulation framework are depicted in Fig. 2a: (i) the *Scenario*, (ii) the *Orchestrator*, (iii) the *Simulator*, and (iv) the *Model Instance*. This figure offers a general overview of the interacting components in a co-simulation framework, while Fig. 2b–d represent the specific implementations of these components inside the three main framework analysed: Mosaik, HELICS and AIOMAS.

The *Scenario* is a representation of the simulated environment that contains the formal knowledge of the entire CPES. It is not an actual physical component of the co-simulation framework. In fact, it represents the configuration offered by the co-simulation framework that manages the startup of the Orchestrator, the initialization of the Simulators, and states the relationships that occur between Model Instances. The *Orchestrator* is the main component of a co-simulation framework and manages the exchange of data from the Simulators and their time regulation and synchronization. *Simulators* instead contain a specific Model Instance class and have different functionalities (e.g. solvers) to perform their domain-specific computations. Simulators instantiate their Models multiple times and govern the resulting collection by acting as a communication adapter with the Orchestrator. In fact, Simulators transmit inputs received by their peers via the Orchestrator and the Orchestrator commands to their Model Instance collection. In return, Simulators receive outputs from Model Instances that are sent to the Orchestrator. Finally, *Model Instances* are representations of multiple homogeneous physical entities. They contain a physical model

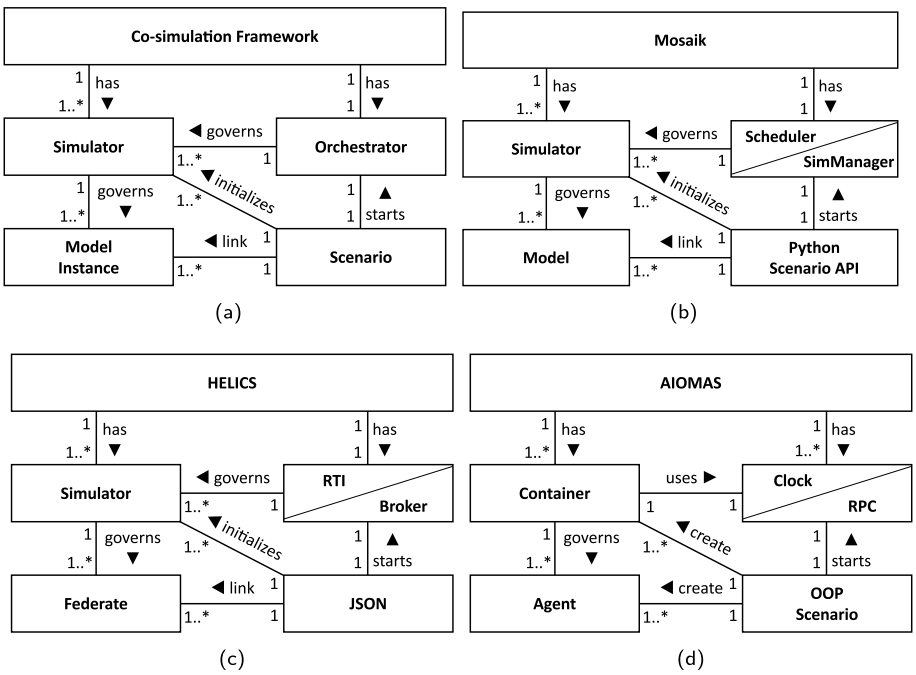

Fig. 2 Component relational schema of a general co-simulation framework (**a**) and its declination for Mosaik (**b**), HELICS (**c**), and AIOMAS (**d**)

that could belong to different mathematical types, ranging from pure algebraic equations to differential equations, as well as finite element methods or behavioural models (Palensky et al. 2017).

In addition, the arrangement of the components addresses three main tasks, which are (i) the *Initialization*, (ii) the *Time Regulation and Synchronization*, and (iii) the *Data Exchange Management*. The *Initialization* task is performed by the Scenario that initiates the Simulators with the proper parameters setting (e.g. time step duration and start date) and communicates the number of Model Instances that compose their collection. The Initialization process finally sets up the co-simulation environment by establishing all the relationships and connections among Model Instances of all Simulators involved in the co-simulation environment. The *Time Regulation and Synchronization* task instead manages and regulates the time step progression of each individual Simulator. In fact, co-simulation can be classified according to its time regulation paradigms (Schweiger et al. 2019), which are: (i) Discrete Event (DE) or event-based regulation, and (ii) Continuous Time (CT) or time-stepped regulation. The DE paradigm proceeds in time by exploiting events that trigger an evolution of the dynamics of the co-simulated environment. Thus, Model Instances communicate via Simulators with each other using events that might change their internal state or trigger other events. Conversely, the CT paradigm determines the evolution of the time step with a constant time interval in which the Simulators evolve their internal states by exchanging inputs and forwarding outputs at the end of each time step. Some co-simulation frameworks are able to handle both paradigms, resulting in a hybrid regulation paradigm. This case requires a complex time regulation algorithm where the synchronization task becomes even more critical. Finally, the *Data Exchange Management* task handles the communication among Model Instances, Simulators, and the Orchestrator by implementing telecommunication protocols that are usually the most effective solution for this task. In data exchange management, the main issue is related to the communication latency that usually affects telecommunication protocols. More specifically, communication latency in co-simulation frameworks refers to the amount of time elapsed from the forwarding of the output variables of one Model Instance to the reception of the variable as input by another Model Instance. Large latency can compromise the overall co-simulation environment when dealing with strict time constraints of a particular Simulator that could internally trigger a time step overflow. In conclusion, the Initialization, Time Regulation and Synchronization, and Data Exchange Management represent the most important challenges in ensuring a reliable, accurate, and stable co-simulation framework.

As previously mentioned the MAS concept could be studied along with co-simulation, indeed we have exploited AIOMAS as follows:

(a) we have used AIOMAS as a modelling library for a MAS simulator that has been integrated into a co-simulation framework. In this case, the co-simulation framework (whichever it is) encapsulates a MAS simulator. We will refer to this concept as 'MAS as a simulator'. (b) We have also used AIOMAS as a tool to build up a co-simulation framework in which the integration of simulators is done through the agent concept. In particular, agents (understood as intelligent entities that communicate with each other and the environment) are designed as wrappers for real external simulators (e.g., building thermal simulators, photovoltaic panel simulator). In

this way, the replication of simulators is done through the spreading and spawning of agents. We will refer to this case as 'MAS as co-simulation'. This type of architecture reflects the possibility of decoupling the agent envelope from the simulator, i.e. the intelligence (e.g., control algorithm, management system) and the physical model respectively. The agent could model explicitly any intelligent control and encapsulate an interchangeable physical simulator (which models the physical behaviour as it is) on which to test intelligent strategies. Indeed, several studies implement co-simulation alike environments exploiting MAS tools (Pipattanasomporn et al. 2009; Roche et al. 2010; Mazzarino et al. 2021; Nunna and Doolla 2012; Jung et al. 2018). The concept of Agents in MAS applications can easily comply with the definition of SoS covering the needs of a co-simulation framework and, in particular, its required components.

The aim of this paper is the scalability analysis of Mosaik and HELICS, two of the most widely adopted co-simulation frameworks in literature. The analysis is performed by applying a comprehensive benchmark of the possible configurations that each framework could implement. In addition, a similar benchmark is introduced for the integration of the AIOMAS library, presenting both MAS as a simulator and MAS as a co-simulation framework [as in Mazzarino et al. (2021)]. In the following sections, the details and peculiarities of these three frameworks are addressed.

Mosaik

Mosaik is a Python co-simulation framework developed to couple existing Simulators in the Smart Grid field. Its general architecture does not preclude other domain applications. Mosaik provides different Application Program Interfaces (APIs) and components for the main functionalities of a co-simulation framework. Firstly, the *Python Scenario API* allows creating a Python script Scenario in which instantiates and establishes input/output relationships between Model Instances and Simulators. The High-level Simulators API instead provides an abstract class with communication, time regulation, and synchronization features already implemented. They are language agnostic, thus allowing the integration of different programming languages (i.e. Python, C++, and JAVA) and Simulator software (e.g. MATLAB). The Low-level API instead offers the possibility to establish a plain network socket for exchanging serialized JSON data to extend Mosaik Simulators integration capabilities. The implementation of this API requires a meta description of the Simulator that states its parameters and the exchanged variables.

Figure 2b depicts the relational entities in Mosaik architecture. The Orchestrator role is fulfilled by two components: the *SimManager* and the *Scheduler*. These two components respectively share the tasks of Data Exchange Management and Time Regulation and Synchronization. The SimManager starts the Simulators that govern their Model Instance collection and, subsequently, handles their data exchange. Mosaik manages multiple Simulators that can create Model Instance collection by instantiating their Models. The Scheduler instead synchronizes the Simulators time regulation and could manage both CT and DE paradigms (only in Mosaik version 3.0 which has integrated the support to DE and allows specification of simulators type).

HELICS

Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS) is a co-simulation framework based on IEEE High-Level Architecture (HLA) standards (IEEE Standard for Modeling and Simulation 2010a, b). It integrates Simulators from different programming languages (i.e. Python, C++, JAVA, Nim) and simulation software (e.g. MATLAB) in a scalable and distributed environment.

The HELICS architecture and its relational entities are presented in Fig. 2c. The Scenario in this framework consists of a JSON configuration file in which all the necessary links and parameters for the instances are made explicit. HELICS introduces a different terminology with respect to Fig. 2a. It retains the concept of Simulators, which in this case, are generic executables that can instantiate a multitude of Federates. Federates represent specific entities defined in the Scenario that executes their respective physical models. HELICS architecture is distributed so each Federate can communicate with others through a publish/subscribe approach (Eugster et al. 2003) via Cores. Cores are components embedded in Simulators that allow their Federates to join Federations and enable communication with the HELICS architecture. The Data Exchange Management task among Federation is guaranteed by the *Broker* component that coordinates the exchange among different Federations. A Broker could also communicate with other Brokers, and consequently with other Federations, enabling the possibility of deploying a hierarchical architecture. Finally, the Orchestrator is managed by the *Run-Time Infrastructure* (RTI), a component inherited by HLA standard, to ensure a proper Time Regulation and Synchronization of the overall co-simulation environment in both CT and DE paradigms.

AIOMAS

AIOMAS is a Python library to implement MAS. It has been chosen to present parallelism between MAS and co-simulation frameworks. At an higher level of abstraction, AIOMAS provides four main classes: (i) the Container, (ii) the Agent , (iii) the Remote Procedure Call (RPC) along with the Clock and (iv) the Object Oriented Programming (OOP) Scenario.

Figure 2d shows AIOMAS relational entities following the generic co-simulation infrastructure described above. The OOP Scenario component in this configuration does not have a specific implementation. In fact, its design and development are completely up to the end user who can decide to create a specific general Python script or distribute Agents linking inside their Python classes (i.e. OOP Scenario). The tasks required to establish the co-simulated environment are similar to the aforementioned frameworks and are: (i) the Container creation, (ii) the Agent collection generation, (iii) and the distributed orchestration infrastructure start-up (i.e. Clocks/RPC). Agents incorporate specific models of the physical entities they describe and execute their behaviour. Containers, on the other hand, host Agents and communicate with them via RPC servers that handle the Data Exchange Management task. In a Container, Agents implement RPC clients that manage remote communication, using the Container as a gateway to reach Agents that belong to other Containers. Each Container implements the task of Time Regulation and Synchronization through the implementation of a shared

distributed Clock. The time evolution of the co-simulation environments could follow the CT or DE paradigms, depending on the user's implementation choice. This peculiarity addresses one of the main challenges of co-simulation, which is the complex time regulation when it comes to hybrid simulation; with AIOMAS, it is possible to distribute the time regulation and customize it at the expense of more implementation effort.

Methodology for benchmarking design

The most complicated and debated issue in co-simulation applications is scalability which is defined as the property of a co-simulation framework to handle an increasing amount of heterogeneous Simulators and their model instances, considering the composite relationships that interconnect them together to run a large-scale complex system, such as a Multi-Energy System (MES). From an Information and Communication Technology (ICT) perspective, scalability is measured typically with three indicators known as scalability dimensions: (i) size, (ii) geographical, and (iii) administrative scalability. Size scalability represents the issues in growing the dimension of the co-simulated system and what are the possible solutions to manage the high number of Simulators and Model Instances to run a huge complex Scenario and its orchestration. Geographical scalability, on the other hand, is the representation of the complexity of managing an increasing number of geographically distributed computational nodes (e.g., different laboratories) to implement a co-simulation Scenario. Finally, administrative scalability represents the difficulties in managing a co-simulation framework when dealing with increasing both previous scalability dimensions, thus the engineering effort required to avoid the complex setup of the co-simulation framework, the orchestrator, and the distribution of Simulators and their Model Instances among network nodes, and their interconnections.

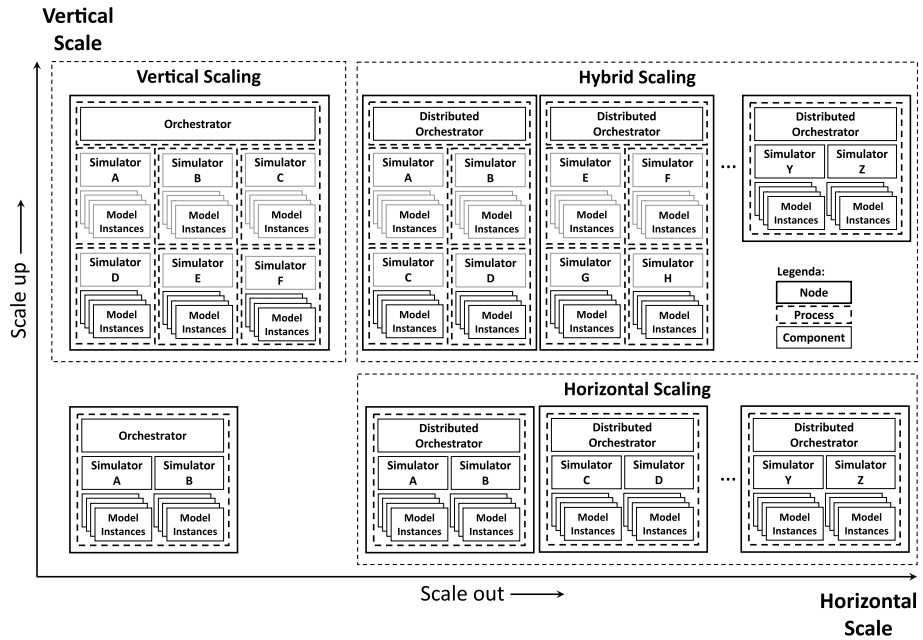

Fig. 3 Vertical and horizontal scaling

The two main scaling directions of a co-simulation framework are (i) vertical scaling and (ii) horizontal scaling, as illustrated in Fig. 3. Vertical scaling takes advantage of the parallel capabilities of a single node to distribute the co-simulated Scenario across multiple processes, each of which runs a certain Simulator. Depending on the Simulators, their Model Instances, and their relationships defined by the Scenario, vertical scaling could be applied with different methods and strategies. It is worth noting that this scaling direction commonly results in limited scaling of the size of the complex system. Conversely, horizontal scaling exploits the distribution of the co-simulation Scenario over multiple network nodes, joining them by means of telecommunication protocols. In this view, different Simulators are distributed over different network nodes that manage their Model Instances. Also, in this case, there are different solutions depending on the relationships between the Model Instances of each involved Simulator. This approach requires a distributed co-simulation Orchestrator that can act as a load balancer that distributes tasks and manages data exchange and synchronization of all working nodes. The above two directions of scalability are not mutually exclusive and, instead, are typically used in a jointed configuration to improve the scalability of a co-simulation framework. Merging Vertical and Horizontal scalability is an advantage when dealing with particular simulation software and/or hardware needed to simulate a specific component of a complex system. For instance, a Digital Real-Time Simulator (DRTS) is required in some specific MES Scenarios to perform an Electromagnetic Transient (EMT) analysis of a power grid (Barbierato et al. 2022). This particular hardware acts as a vertical scaling component of the jointed scaling vision to enable fast real-time simulation of the power grid model. Then, the DRTS will be interconnected with a distributed co-simulation environment running other MES models. This distributed configuration participates in the hybrid scaling vision of implementing horizontal scaling.

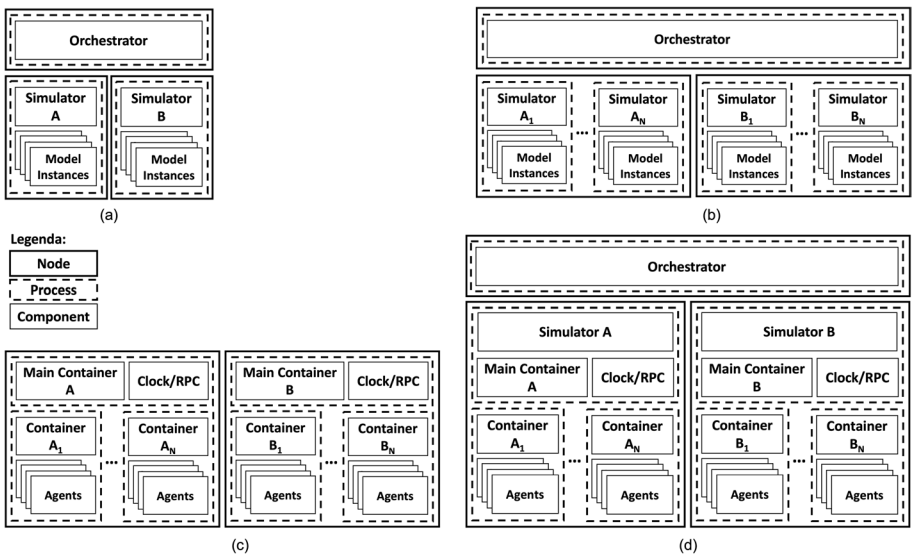

Fig. 4 The proposed co-simulation benchmark configurations: (**a**) Classic Co-simulation, (**b**) Multi-process Co-simulation, (**c**) Multi-Agent System as Co-simulation framework, and (**d**) Classic Co-simulation configuration with encapsulated multi-process Multi-Agent System

For the purpose of evaluating the scalability of the co-simulation frameworks presented in "Enabling technologies for co-simulation environments" section, a hybrid scaling approach has been chosen among the three possible options to assess what could be the impact of scaling up and scaling out a generic MES Scenario on a distributed cluster of nodes. The benchmark configurations in Fig. 4 are described in the following.

(a) The *Classic Co-simulation configuration* (see Fig. 4a) is the common configuration of co-simulation frameworks (i.e. Mosaik and HELICS) where Simulators are run by different cluster nodes handled by the Orchestrator master node that manages their data exchange and synchronization. Each Simulator node manages iteratively its M Model Instances in a single process. By implementing a distributed deployment of Simulator, this configuration will enhance the simulation capabilities with respect to standalone simulation. However, performances are expected to be low because each cluster node runs the assigned simulator and its instances in a single iterative process. It is worth noting the significant impact of the information exchange among Simulators and the Orchestrator.

(b) The *Multi-process Co-simulation configuration* (see Fig. 4b) evolves the classic configuration by enabling a multi-process division of a Simulator node (e.g. Simulator A), replicating it in N Simulator processes (e.g. Simulator $A_1, ..., A_N$). Considering M Model Instances, each Simulator process manages M/N Model Instances. With respect to Classic Co-simulation configuration, it enables a multi-process execution of the assigned simulator and its instances for each cluster node. This configuration considerably raises the performance of each simulator time step execution. However, the setup of the multi-process execution may take longer and could be a drawback that reduces the configuration performances. The information exchange among Simulators and the Orchestrator instead is identical to the previous configuration.

(c) The *Multi-Agent System as Co-simulation configuration* (see Fig. 4c) is the typical configuration of the AIOMAS framework in which agents are represented as simulators. When dealing with a small number of Agents, AIOMAS exploits the Main Container for each Agent class that is spread on one of the available cluster nodes. Each Main Container manages (i) the data exchange with its fellow and its Agents through the RPC protocol and (ii) the distributed synchronization through its internal Clock. Likewise Model Instances, Agents (Simulators) are replicated in a single process by applying a concurrent multi-threading. When dealing with a high number M of Agents, a Main Container (e.g. Main Container A) could delegate to N spawned Containers (e.g. Container $A_1, ..., A_N$) the Agent management, resulting in M/N Agents assigned to each child Container. This configuration avoids the required information exchange for Time Regulation, Synchronization, and Data Exchange Management of the Classic and Multi-process Configurations. In fact, the Time Synchronization and Regulation is distributed among the Main Container via the Clock/RPC. Moreover, each Agent directly communicates its output (i.e. events) with other Agents interested in receiving it. This approach enhances the performances with respect to having a centralized Data Exchange Management with an Orchestrator. The drawback of this configuration is related to the setup of the N spawned Containers when dealing with a high number of Agents.

(d) The *Classic Co-simulation configuration with encapsulated multi-process Multi-Agent Systems* (see Fig. 4d) manages a hybrid configuration of the Classic Co-simulation

configuration where each Simulator is a MAS simulator. They are built with AIOMAS Main Containers that spawn N child Container in different sub-processes. Each of the child Containers manages M/N Agents, enhancing the scalability of a Classic Co-simulation configuration by integrating MAS simulators that manage several physical entities exploiting multi-process and multi-threading AIOMAS capabilities. This configuration exploits the capabilities of spawning Containers of the Multi-Agent System as Co-simulation configuration without incurring typical drawbacks of the Multi-Process Co-simulation configuration, raising the performance of this configuration. However, this configuration could suffer from the typical information exchange drawback of the Classic Co-simulation configuration due to the central Data Exchange Management of the Orchestrator.

Benchmark key performance index (KPI)

The KPI is defined over a time interval of the main contributions that compose the *Total Execution Time* of a co-simulated Scenario. These processes are depicted in Fig. 5 and are: (i) the *Scenario Setup Process*, in which the co-simulation framework starts the Orchestrator, initializes Simulators with their Model Instances, and, finally, links all the Model Instances to deploy the co-simulated Scenario; (ii) the *Co-simulation Process* that is an iterative process in which the co-simulated Scenario evolves its state each Time Step (i.e. its fundamental unit); (iii) the *Termination Process* in which the co-simulation framework stops the Orchestrator, releases Model Instances and terminates Simulators execution. Each Time Step is a complex routine in which each Simulator retrieves the input dependencies for its Model Instance collection, iteratively executes each Model Instance calculation updating its state, and, finally, collects Model Instance collection outputs to forward them to other Simulators. Simulators operate in parallel as depicted

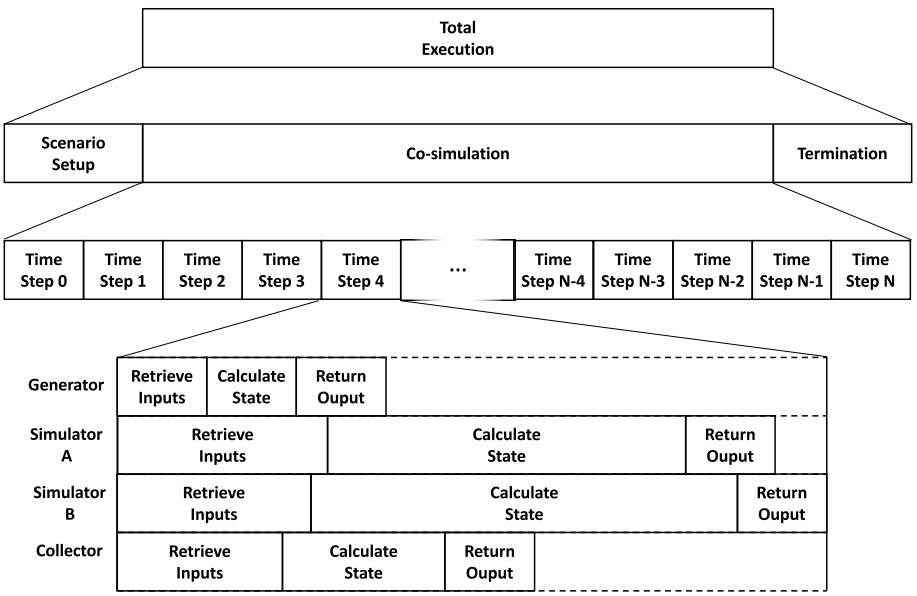

Fig. 5 Decomposition of the Total Execution Time in its main contribution: (i) the Scenario Setup, (ii) the Co-simulation, and (iii) the Termination Processes. The former Co-simulation Process could be decomposed into its main time interval units, the Time Steps

in Fig. 5 where the execution of the four Simulators is highlighted for the *Time-Step 4*. This parallel execution impacts the accuracy of the co-simulated solution with a finite time step latency related to Simulator input/output dependencies that causes negligible inaccuracies of the solution with respect to a standalone simulation (Steinbrink et al. 2018). Each Time Step duration could vary depending on the input/output Data Exchange Management, the communication latencies, and from the particular computational condition of each cluster node.

Three main time-based KPIs have been employed to evaluate the scalability of each benchmark configuration and its Scenario implementation:

- the *Setup Execution Time* which is the time interval in which the co-simulation framework executes the Scenario operations;
- the *Average Time Step Duration* μ_T that is estimated as the maximum of the means of the time duration T of the S co-simulative time steps of each Simulator Sim_i involved in the co-simulation environment I;

$$\mu_T = \max_{i \in I} \mu_T^i \quad where \quad \mu_T^i = \frac{\sum_{n=1}^{S} T_n^i}{S}, \quad I = \{Sim_1, ..., Sim_M\} \tag{1}$$

- the *Total Execution Time* that includes all the contributions of the Scenario Setup, the Co-simulation, and the Termination Processes.

The main objective of this paper is to understand which of the proposed benchmark setups performs better in terms of simulation time when it comes to increase the size of the simulated environment. The chosen KPIs are general and reflect the main simulation times, so they are useful for making comparisons on the scalability of different frameworks.

Setup of co-simulation scenario

In order to test the benchmark presented in "Methodology for benchmarking design" section, a common realistic scenario has been prepared. The chosen simulators and related physical models are heterogeneous ranging from very simple to more complex. Only the simplest interactions and data exchanges have been included (same time loop, feedback exchanges, or control actions have been excluded) in order to keep the simulation workflow as simple and linear as possible. The motivation behind this choice is to be able to fully relate simulation time performances to the increase of running instances, ignoring any other possible slowing down aspect.

A high-level representation of the multi-model energetic scenario is presented in Fig. 6, while Fig. 7 depicts the declination of the benchmark setup to the specific case study. The energetic Scenario includes four typical actors of a networked urban environment. The four Simulators (or Containers in the AIOMAS perspective) are: (i) the Meteo Simulator, (ii) the PV Simulator, (iii) the Building Simulator, and (iv) the Power Grid Simulator. Furthermore, still looking at Fig. 6 the simple input/output interactions between Simulators are presented. Specifically, both the PV and the Building Simulators take weather data from the Meteo Simulator as inputs to perform their calculations. Their outputs are sent to the Power Grid Simulator which uses them to calculate the

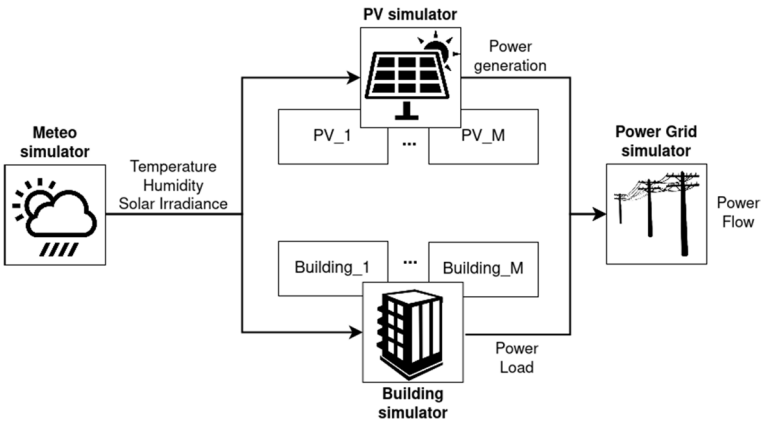

Fig. 6 Simulator of the analyzed physical MES Scenario

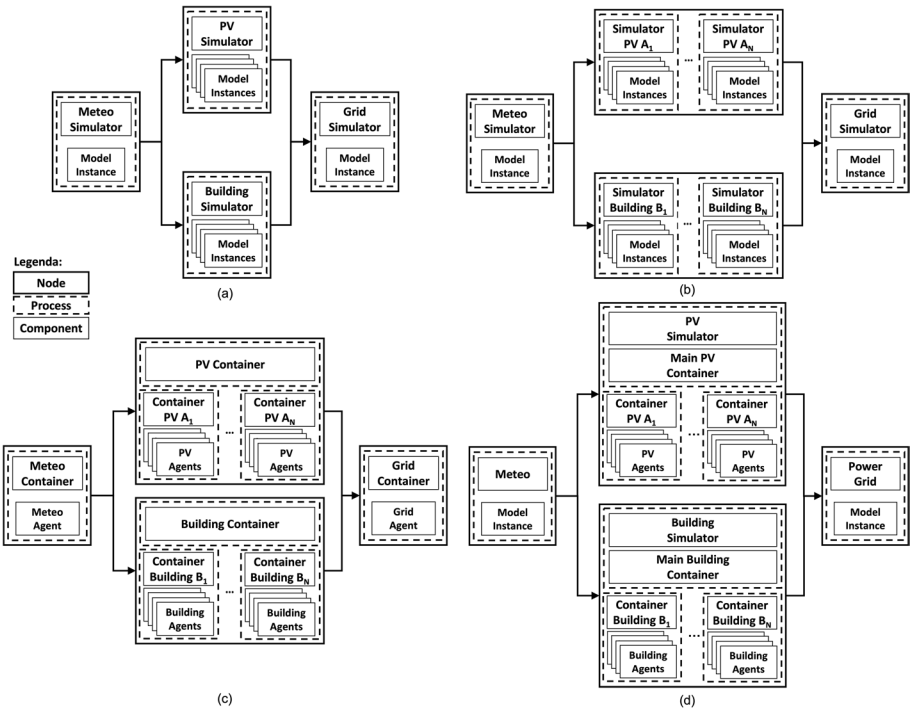

Fig. 7 Different implementations of the co-simulation framework benchmark configurations: (**a**) Classic Co-simulation, (**b**) Multi-process co-simulation, (**c**) MAS as co-simulation, and (**d**) Classic Co-simulation with encapsulated MAS

power flow of the power grid. The time resolution of the simulators is configurable and could differ from one simulator to another. All of these simulators have really short wall-clock times for executing a single step thus the simulation time-step could range from seconds to hours. Nevertheless, the time resolution choice must take into consideration the physical significance and constraints of the involved model, so for the Building simulators, the chosen simulation time-step was 1 h. In contrast, mainly to show the possibility of mismatched time resolution, photovoltaic simulators update their state every 15 min. Each Simulator is presented in the following sections.

The meteo simulator

The Meteo Simulator is a stateless Simulator and acts as a weather file reader (.epw format), obtaining measurements about the weather (i.e. temperature, solar irradiance, and humidity) for each simulation time step. It forwards these outputs to both PV and Building Simulators.

The PV simulator

The PV Simulator is a stateless Simulator that uses the *estimate PV energy* service of the model in Bottaccioli et al. (2017). It estimates the hourly energy production of PV solar panels for each simulation time step. The given model receives weather conditions and information about the surface area of each panel and uses them to perform its calculations. It also estimates the cell temperature using the so-called *NOCT* method (Brihmat and Mekhtoub 2014) when wind speed is not available as input and, on the other hand, it uses the *Mattei* method (Mattei et al. 2006) when wind speed is available. The output of this simulator is the generating power for the given time step, which is forwarded to the Power Grid Simulator.

The building simulator

The building simulator is a stateful Simulator that exploits the model presented in Mazzarino et al. (2021) to simulate a building equipped with a heat pump system. Within the model, the thermal behaviour of the envelope is treated with a Resistance-Capacitance model (Massano et al. 2019) and a fine-grained model of the heat pump with all subsystems (e.g., emission, distribution, and generation) is provided. The model calculates the hourly heating demand of the building considering also the heat contributions generated from human occupancy and appliances through archetypes patterns. The heat load is then converted to the heat pump power request. To conclude, the former model has four main functionalities that calculate: (i) the real-time heat pump power demand, (ii) the heat pump power demand forecasting, (iii) the heat pump energy flexibility, and (iv) the actuation commands of the heat pump control strategy. The proposed Scenario only uses the heat pump's real-time power demand as output for the Power Grid Simulator. The Building Simulator is stateful because it keeps track of the previous indoor air temperature condition when calculating the heat pump demand for a new time step. The inputs for this simulator are the weather information sent by the Meteo Simulator, while the output is the heat-pump demand load for the given time step, which is forwarded to the Power Grid Simulator.

The power grid simulator

The power grid simulator is a stateless Simulator that emulates a power grid model with different connected loads. The power grid is based on the model presented in Estebsari et al. (2021) and takes advantage of the Python electrical system analysis tool pandapower (Thurner et al. 2018). This model contains a representation of the network topology and its main components (e.g. buses, lines, transformers) along with their nominal parameters. It is easily configurable and allows the integration of loads, generators, and storage. This Simulator collects the hourly data coming from the PV Simulator and the Building Simulator (i.e. the PV power generation and the Building power load)

to solve the power flow analysis of the grid network. In this simulation, it is actually used as a discrete event simulator that is triggered by the reception of both building and PV data.

Benchmark configuration

The benchmark design will consider the general Scenario depicted above. The scalability problem is addressed by rising up the M number of Model Instances (i.e. Agents for AIOMAS framework) of the PV and Building Simulators. The benchmark will run on an Internet distributed computing system, which is composed by a cluster of 4 nodes with a master node serving as Orchestrator when needed. The implementations of the Scenario for each configuration and its cluster deployment are described in Fig. 7 and are:

(a) The *Classic Co-simulation implementation* (see Fig. 7a) that uses the master node as Orchestrator and the four cluster nodes, one for each of the above-mentioned Simulators. Two cluster nodes manage respectively the Meteo Simulator and the Power Grid Simulator, each one handling its single Model Instance. The remaining nodes manage respectively the PV Simulator and the Building Simulator. Finally, each Simulator iteratively runs M Model Instances in its process;

(b) The *Multi-process Co-simulation implementation* (see Fig. 7b) that uses the same implementation of the previous case (a). However, the cluster nodes, assigned to PV and Building Simulators, replicate on N processes each individual Simulator, equally distributing in each process the M Model Instances. For instance, each of the PV Simulator processes $A_1, ..., A_N$ manages M/N Model Instances;

(c) The *Multi-Agent System as co-simulation implementation* (see Fig. 7c) that uses the four cluster nodes without the master node. Two cluster nodes implement a simple Container structure that handles respectively the Meteo Agent and the Power Grid Agent. The other two cluster nodes manage respectively the Main PV Container and Main Building Container, each one handling M Agents. Each Main Container spawns N child Containers that handle M/N Agents. For instance, the Main PV Container spawns child Containers $A_1, ..., A_N$ each one managing M/N equally distributed Agents;

(d) The *Classic Co-simulation implementation with encapsulated Multi-Process Multi-Agent Systems* (see Fig. 7d) that uses the same implementation of the Classic Co-simulation case. However, the PV and Building Simulators processes on the two cluster nodes encapsulate the Main Container class that manages the spawning of N child Containers in different sub-processes. Each of the child Container manages M/N Agents like in the Classic Multi-Agent System implementation of Main PV/Building Containers;

Finally, the analysis was conducted on seven different co-simulation configurations. For each configuration, the M number of Model Instances (PVs and buildings) was scaled from 1, 100, 10k, 100k, to 1M with the exception of configurations involving Mosaik. In fact, Mosaik has a design limitation on the maximum size of data exchanged that did not allow scaling beyond 10k of Model Instances. It is correct to point out that this limitation could be overcome by applying some modifications to the inner SimPy (python library) methods used in the Mosaik source code. However, since these modifications are not user-friendly and are not included in Mosaik's API implementation, this framework was used as is, with this scaling limitation. The proposed configurations are: (i) two configurations of the Classic Co-simulation (Fig. 7a) for HELICS and Mosaik; (ii)

Table 1 Summary of the tested configurations

Tested configurations	description
Mosaik	Classic Mosaik
HELICS	Classic HELICS
AIOMAS	Co-simulation framework implemented through AIOMAS library
Mosaik multi-process	Mosaik implementation parallelized through multiprocessing
HELICS multi-process	HELICS implementation parallelized through multiprocessing
Mosaik-AIOMAS	Mosaik co-simulation with encapsulated MAS simulator (AIOMAS)
HELICS-AIOMAS	HELICS co-simulation with encapsulated MAS simulator (AIOMAS)

Table 2 Summary of the computational nodes used in the simulations

Node name	Core	RAM	Storage	OS	Role
Cloud Master	16	64 Gb	128 Gb	Ubuntu server 20.04.2 LTS	Orchestrator (when needed)
Cloud 1	32	128 Gb	256 Gb	Ubuntu server 20.04.2 LTS	Building Simulators
Cloud 2	32	128 Gb	256 Gb	Ubuntu server 20.04.2 LTS	PV Simulators
Cloud 3	32	128 Gb	256 Gb	Ubuntu server 20.04.2 LTS	Meteo Simulator
Cloud 4	32	128 Gb	256 Gb	Ubuntu server 20.04.2 LTS	Power Grid Simulator

two configurations of the Multi-process Co-simulation (Figure 7b) for HELICS multi-process and Mosaik multi-process; (iii) one configuration of MAS as a Co-simulation framework (Fig. 7c) for AIOMAS; (iv) two configurations of the Classic Co-simulation with encapsulated MAS (Fig. 7d) for HELICS-AIOMAS and Mosaik-AIOMAS. To conclude and easily interpret the analysis of experimental results, in the next section, a summary of all the tested configurations is presented in Table 1.

Experimental results

The co-simulation benchmark has been deployed in a Virtual Machines environment based upon OpenStack cluster with the specification illustrated in Table 2.

To test the scalability of the different implementations presented in Fig. 7, the Multi-model energetic Scenario has been used as a baseline for the seven proposed configurations in "Benchmark configuration" section. The co-simulated environment has been run for seven winter days with configurable time resolutions for each simulator (1 h for buildings and power grid, 15 min for PVs and meteo). Model Instances of Building and PV Simulators were scaled up. In the next sections, the time-based KPIs mentioned in "Benchmark key performance index (KPI)" section are presented to evaluate the comparison among the different co-simulation framework configurations.

Setup execution time

The setup execution time considers the time duration of the Scenario configuration operations. In a nutshell, these operations are related to the Initialization task of a general co-simulation framework. In Fig. 8, the setup execution time performances of the different configurations are presented. Classic Mosaik keeps the setup time nearly constant around 7s but it cannot handle more than 10k instances due to its design limitations. This result highlights that Classic Mosaik is a really well-designed and efficient

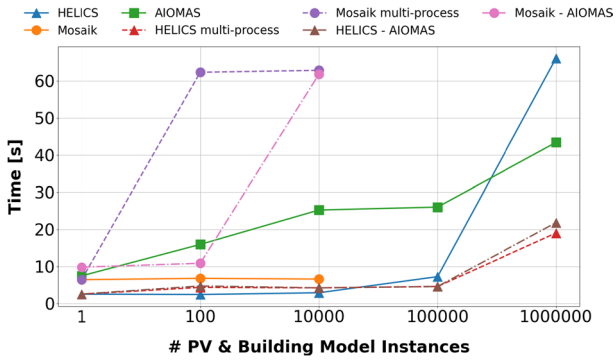

Fig. 8 Setup time duration of the different co-simulation frameworks and their configurations

framework when dealing with limited-size co-simulation environments. Mosaik multi-process instead presents the most time-consuming initialization phase among the benchmarked frameworks. In fact, Mosaik multi-process experiences a big setup time duration difference when rising from 1 instance (i.e. 1 process) to 100 instances (i.e. multiple processes), going from around 6s to 62s. Then, the trend flattens out because the same number of parallel processes have been launched. In this graph HELICS, HELICS multi-process, and HELICS-AIOMAS do not differ much in performance, but an interesting aspect of multi-process HELICS is that it slightly reduces its computation time when going from 100 to 10k model instances, saturating the available cores. Classic HELICS suffers a drastic increase in the setup time from around 7s to 66s only when the instance number passes from 100k to 1M. The best performing configuration is HELICS multi-process framework which is stable at around 5s from 1 to 100k instances. It is worth noting that this framework drastically reduces the time increase of classic HELICS when going from 100k to 1M instances by distributing the computational power for any Simulator over multiple cores, resulting in a setup time of 19s. For the Mosaik-AIOMAS framework, results are similar to the Mosaik multi-process framework with the main difference that the major time increment occurs when going from 10s for 100 instances to 61s for 10k instances. This is due to the threshold used to decide when AIOMAS starts the creation of the multi-process containers. In our case, this threshold is above 100 Model instances. This behaviour does not occur when coupling HELICS and AIOMAS because launching multi-processes and container creation is handled by HELICS with less computational overhead, following the same HELICS multi-process trend.

Average time step duration

The average time step duration reports the execution time of a complete time step that is estimated as the maximum among the mean time step duration of every single Simulator. Figure 9 reports this KPI for all the benchmarked co-simulation frameworks. The average time step duration differences among the co-simulation frameworks are really small up to 10k instances, except for Mosaik-AIOMAS and AIOMAS due to the overhead introduced by the message exchange between Agents. Indeed, Mosaik-AIOMAS takes 0.78s at 10k instances and AIOMAS takes 0.76s, while all the others take around 0.1s. As already mentioned, a comparison with Mosaik is only possible below 10k Model Instances. Up to this point, the benefits of a multi-processing approach are

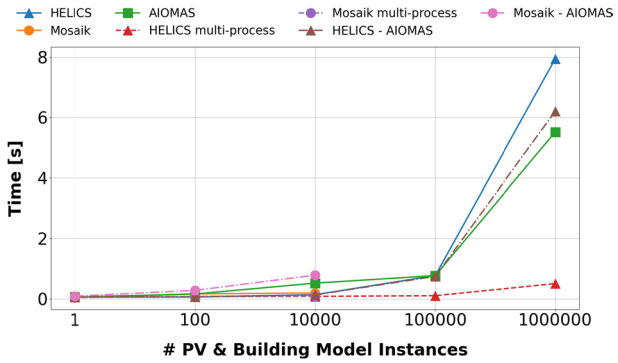

Fig. 9 Average time step duration of the different co-simulation frameworks and their configurations

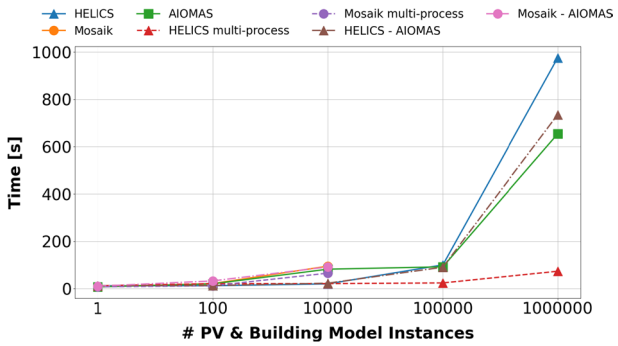

Fig. 10 Total execution time of the proposed Scenario for the different co-simulation frameworks and their configurations

not significant compared to the respective classic solutions. The limitations of both classic and the AIOMAS frameworks are clearly visible when dealing with more than 10k Model Instances. In fact, classic HELICS takes about 16 times longer than its multi-process version with 1M Model Instances. Instead, solutions incorporating AIOMAS even though they perform better than pure HELICS are outperformed by HELICS multi-process. Overall, the configuration with multi-process HELICS performs the best, particularly with paramount performance on the largest scaling Scenario.

Total execution time

Total execution time represents the time duration of the entire co-simulation process from Initialization to the end of all the tasks. Figure 10 presents trend similarities with the average time step duration KPI in Fig. 9.

In fact, this KPI is a composition of the setup execution time and the sum of the duration of the time steps required to fulfil the co-simulation of the Scenario. When dealing with short simulations, the effect of the setup execution time on the total execution time is larger. Instead, with longer simulations, this effect could be negligible. The KPI trends of the total execution time in Fig. 10 consolidate the performance considerations made in "Average time step duration" section, showing the HELICS multi-process as the best performing framework.

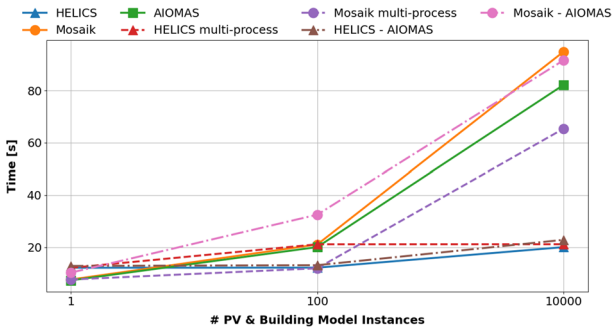

Fig. 11 Zoom on the first 10,000 instances for the Total execution time

Table 3 Qualitative comparison among Mosaik, HELICS, and AIOMAS

Co-simulation frameworks		Mosaik	HELICS	AIOMAS
Scenario	Configuration	Programmatic (scripting)	JSON	Programmatic (Agent-based/ scripting)
	Complexity	Low	Low	High
Orchestrator	Synchronization	Scheduler (SimPy)	Scheduler (RTI)	Custom (Distributed clocks)
	Communication paradigm	Request/ Response	Publish/ Subscribe	Request/ Response
	Data Exchange	TCP	ZeroMQ	TCP/RPC
	Tipology	Time-stepped/ Event-based	Time-stepped/ Event-based	Time-stepped/ Event-based
Simulator Integration	Programming languages / Simulator Software	Python MATLAB Java C++	Python MATLAB Java C++ Nim	Python
	Integration Complexity	Low	Low	High
Scalability	Horizontal	Distributed	Distributed	Distributed
	Vertical	Multi-process (Manual)	Multi-process (Automatic)	Concurrent Multi-threading (Automatic) Multi-processing (Manual)
	Performance	Low	High	Medium

Figure 11 is an enlarged representation of Fig. 10, that focuses on the first 10k Model Instances so as to fairly compare the execution times of all proposed solutions. By analyzing this graph, it can be seen that Mosaik and Mosaik-AIOMAS diverge about 80s while Mosaik multi-process by about 50s from the best performing solutions. This is due to the higher setup execution time of these configurations and greater variability of the average time step execution.

Qualitative comparison

Table 3 presents a qualitative comparison among the benchmarked co-simulation solutions. This comparison is done considering the only multi-process versions of HELICS and Mosaik frameworks because their classic versions, as well as the hybrid configurations (HELICS-AIOMAS and Mosaik-AIOMAS), do not bring any KPI improvements. In addition, also AIOMAS is included in this comparison, it is not a co-simulation

framework but, as in our case, it could be used as a viable alternative to build one from scratch. The qualitative comparison among the benchmarked co-simulation solutions presented in Table 3 points out different characteristics of the frameworks that can help in choosing the right solution depending on the Scenario under analysis. Concerning the Scenario component (see Scenario in Table 3), a first consideration could be made on how the configuration is performed and what effort is required to fulfill the Scenario implementation. Mosaik needs a Python script in which the end user must start the Simulators, instantiate their Models and link them through connectors. This process requires low effort since it is well documented and standardized. HELICS instead uses a JSON file to set up all the required configurations and, thus, has a really low implementation complexity. However, this process could suffer from errors in deploying the correct connection among Model Instances. Finally, AIOMAS requires more effort in the initialization due to the absence of predefined Scenario standards. Linking and instantiating the Models can occur within the Agent definitions or in separate scripts, and is completely up to the end user design. This results in a more complex but freer implementation process.

Simulation management, which is the primary responsibility of the Orchestrator, has different characteristics among the three frameworks (see Orchestrator in Table 3); specifically with respect to communications and synchronization. Mosaik handles synchronization via its Scheduler, limiting or making cumbersome custom intra-step operations. Vice versa, HELICS offers an RTI to manage the time regulation and synchronization, allowing greater freedom of development than the Mosaik Scheduler. In contrast, AIOMAS does not offer a synchronization orchestrator, thus time regulation must be autonomously implemented by exploiting distributed external clocks and designing the wrapping agents properly. This means that the end user must take into account this task in a programmatic way encapsulating time management into Agents or creating an external agent as an orchestrator. The communication approach is based on Request/Response for both Mosaik and AIOMAS that exploit respectively TCP and RPC over TCP for communication purposes. HELICS instead uses a Publish/Subscribe approach that could enhance the performance but can lead to wasting resources in data polling requests. To conclude, the three frameworks implement both event-based and time-based simulation paradigms.

Another important aspect is Simulator Integration (see Simulator Integration in Table 3). While HELICS and Mosaik allow the integration of a wide range of Simulators based on different programming languages and simulation software, AIOMAS has no particular API for integrating other programming languages than Python and, if necessary, this must be implemented ad-hoc by the end user. Therefore, the integration of the Simulators in Mosaik and HELICS results in a low-effort task. AIOMAS instead is more complex.

Looking at Scalability (see Scalability in Table 3), the three frameworks can deploy Simulators or Containers horizontally by distributing them on different cluster nodes for their management. However, they differ in the possible implementation of vertical scaling. For instance, Mosaik allows multi-processing but its implementation has to be done manually. HELICS multi-processing instead is handled automatically by the co-simulation framework. AIOMAS provides an automatic concurrent multi-threading

of Containers. Conversely, AIOMAS multi-processing requires a manual implementation. Finally, the above features and proposed results allow for a comparison of the overall scalability performance. Mosaik performed the worst scalability mainly due to the intrinsic limitation that does not allow to scale up beyond 10k Model Instances. AIOMAS performed medium scalability limited by the overhead increment when dealing with a rise of Agents. HELICS returned the best scalability, showing that its multi-process management of Simulators is capable of reaching 1M instances with paramount performances.

Discussion on experimental results

The experimental results presented in previous Sections allows to determine the proper selection among the presented co-simulation frameworks and configurations depending on the MES scenario dimension:

(a) Mosaik with a Classic configuration offers easy implementation of a MES scenario and is a well-documented framework. It works well with small to medium scenarios but has limited scalability when going over large scenarios. Furthermore, when dealing with complex relationships and data exchange among simulators it is not as efficient as other solutions.

(b) HELICS with a Classic configuration also offers extensive documentation and an easy implementation, a little less intuitive with respect to Mosaik. It easily scales over a million instances, being an optimal choice for small to large scenarios. Thanks to its publish/subscribe paradigm (Eugster et al. 2003), complex interactions among simulators are easier to implement.

(c) AIOMAS can be used to implement a co-simulation framework, but, being a Generic Python library for MAS, the implementation is up to the user. Thus, the documentation of AIOMAS is not oriented to this kind of usage resulting in a lack of support. Depending on these reasons it must require much more effort and could not be suitable for inexperienced programmers. On the other side, it gives a lot of freedom when designing interactions among agents or complex data exchange workflows. It is suitable for highly customized scenarios such as event-based Simulators that behave in a sequential workflow. It can scale from small to large scenarios but with significantly lower performances with respect to HELICS due to the high overhead of the Agent Data Exchange Management.

(d) Mosaik with a Multi-processing configuration is thought to speed up the Mosaik Classic configuration. The implementation requires some manual workarounds that are not embedded into the Mosaik framework as it is. It can only manage small/medium scenarios because it suffers from the same limitations of the classic version, but it brings some benefits in terms of timing performance. Then choosing this configuration will depend on a personal trade-off between implementation efforts and slightly better timing performances. As a last consideration it could be useful when dealing with simulators running on different machines.

(e) HELICS with a Multi-processing configuration is the best performing solution in terms of scalability. It can support above the million Model Instances with stunning timing performances. Thus it is suitable for very large scenarios, or very complex systems populated by lots of modelled components to be. The multi-processing configuration is

well described and embedded in the HELICS documentation. From the point of view of Simulators interactions have the same qualities of classic HELICS.

(f) Mosaik with encapsulated AIOMAS configuration integrates the MAS simulator inside the Mosaik framework. The implementation of this configuration is not straightforward but some examples in Mosaik documentation are present. By looking at the timing performance when scaling up it does not bring any benefits with respect to the Mosaik or AIOMAS classic solutions, and it suffers from the same model instance limit of Mosaik. Thus, it is suitable for small/medium scenarios. In addition, it allows encapsulation of a MAS ecosystem inside a co-simulation framework. This can be useful when dealing with complex systems having a high number of interactions that we want to contain inside the MAS environment.

(g) HELICS with encapsulated AIOMAS configuration integrates the MAS simulator inside the HELICS framework. No documentation for this configuration is present, making its implementation require more effort. In a very large scenario, it performs slightly better than HELICS with Classic configuration but it cannot be compared with the performances of multi-processing HELICS. The last consideration is that the reasoning on interactions above mentioned for Mosaik-AIOMAS still holds for this configuration, making HELICS-AIOMAS the best choice among the solutions with encapsulated AIOMAS.

In conclusion, up to 100 Model Instances for each Simulator, Mosaik with a Multi-processing configuration, HELICS with a Classic configuration, and HELICS with encapsulated AIOMAS configuration are the best performing solutions. The choice could be dictated by the needs of the analysis: Mosaik has a very simple implementation, but places some constraints on interactions between simulators, HELICS offers more freedom on interactions, and AIOMAS offers much more freedom, but with a greater implementation effort. By increasing the number of instances, especially from 10 k, the best solution in terms of time performance and with the same features is the HELICS with a Multi-processing configuration.

Conclusion

This paper compared the scalability performance of two popular co-simulation frameworks and a MAS library used to build a co-simulation framework. Mosaik, HELICS, and AIOMAS were tested on a simple multi-model energy scenario in order to understand their ability to increase the number of co-simulation instances. The increase in model instances, which might be excessive for the given scenario, was used as a way to understand the response of these solutions when dealing with larger scenarios, from the perspective of simulating entire cities or regions. The co-simulation scenario consisted of a weather simulator, a photovoltaic simulator, a building heating/cooling simulator, and a power grid simulator. The complexity of the models involved was heterogeneous, but the interactions between them were kept simple. The reason for this choice was to be able to easily increase the number of model instances for the PV and Building simulators and to focus the study on the increasing size of the problem. The results show that Mosaik has a limitation in the number of connected entities that stopped at 10k model instances of PV and buildings. On the other hand, HELICS demonstrated the ability to scale up to 1M model instances. AIOMAS demonstrated

the same ability to scale as HELICS with respect to the number of instances. However, HELICS outperforms AIOMAS in terms of simulation time if the simulators are run in multiprocess. To draw some conclusions, it is possible to say that Mosaik is a very useful framework when dealing with small/medium size environments, offering good quality results and ease of implementation. AIOMAS allows implementing co-simulation for very large scenarios, but it does not scale perfectly because of its exponential increase in overhead. Finally, HELICS outperforms all other solutions in terms of scalability while retaining all the advantages of a well-structured framework like Mosaik. HELICS, in particular when implemented in its multi-processing configuration, gives the best scalability performance allowing to be suitable for any scenario size, furthermore, it is easy to be implemented and allows more freedom on the interactions design due to its publish subscribe paradigm, on the overall this configuration proved to be the most flexible and performing above all the tested ones. This work is intended to test and compare the co-simulation frameworks in different configurations on their basic ability to scale up a generic MES scenario. So, the time-based KPIs are still related to the kind of MES scenario and simulators involved. Thus, a future works will be the standardization of benchmarking standards for co-simulation framework taking into consideration the different kinds of simulators characteristics (e.g. DE vs. CT, real-time vs. non real-time). Moreover, the development of co-simulation scenarios in cluster computing systems by integrating simulators and co-simulation frameworks in containers, such as Dockers managed by a Kubernetes orchestrator, is part of our future work to study the scalability of large co-simulation setups.

Abbreviations

DE	Discrete event
DER	Distributed energy resources
DRTS	Digital real-time simulator
EMT	Electromagnetic transient
FMI	Functional mock-up interface
HLA	High-level architecture
ICT	Information communication & technology
KPI	Key performance indicator
MAS	Multi-agent system
MES	Multi-energy system
OOP	Object oriented programming
PV	Photovoltaic
RES	Renewable energy source
RPC	Remote procedure call
RTI	Run-time infrastructure
SoS	System of systems

Acknowledgements
Not applicable.

About this supplement
This article has been published as part of *Energy Informatics Volume5 Supplement 4, 2022: Proceedings of the Energy Informatics. Academy Conference 2022 (EI.A 2022)*. The full contents of the supplement are available online at https://energyinformatics.springeropen.com/articles/supplements/volume-5-supplement-4.

Author contributions
LB: Methodology, Supervision, Writing; PRM: Simulation models, Writing; MM: Software development; EP: Methodology, Supervision, writing; LB: Conceptualization, Supervision, Methodology, Writing. All authors read and approved the final manuscript.

Funding
This paper is funded by the authors. Affiliation: Politecnico di Torino.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Published: 21 December 2022

References

Abgottspon H, Schumann R, Epiney L, Werlen K (2018) Scaling: managing a large number of distributed battery energy storage systems. Energy Inf 1(1):55–71
Authors U (2021) Glasgow Climate Pact. Accessed 26 Jun 2022. Available from: https://unfccc.int/documents/310475
Barbierato L, Estebsari A, Bottaccioli L, Macii E, Patti E (2020) A distributed multimodel cosimulation platform to assess general purpose services in smart grids. IEEE Trans Ind Appl 56(5):5613–5624
Barbierato L, Schiera DS, Patti E, Macii E, Pons E, Bompard EF, et al (2020) GAMES: a general-purpose architectural model for multi-energy system engineering applications. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC); p. 1405–1410
Barbierato L, Pons E, Mazza A, Bompard E, Subramaniam Rajkumar V, Palensky P et al (2022) Stability and accuracy analysis of a distributed digital real-time co-simulation infrastructure. IEEE Transactions on Industry Applications. p. 1–1
Bhattarai BP, Lévesque M, Bak-Jensen B, Pillai JR, Maier M, Tipper D et al (2016) Design and cosimulation of hierarchical architecture for demand response control and coordination. IEEE Trans Industr Inf 13(4):1806–1816
Blochwitz T, Otter M, Arnold M, Bausch C, Clauß C, Elmqvist H, et al (2011) The functional mockup interface for tool independent exchange of simulation models. In: Proceedings of the 8th international Modelica conference. Linköping University Press; p. 105–114
Bottaccioli L, Estebsari A, Pons E, Bompard E, Macii E, Patti E et al (2017) A flexible distributed infrastructure for real-time co-simulations in smart grids. IEEE Trans Industr Inf 13(6):3265–3274
Bottaccioli L, Patti E, Macii E, Acquaviva A (2017) GIS-based software infrastructure to model PV generation in fine-grained spatio-temporal domain. IEEE Syst J 12(3):2832–2841
Brihmat F, Mekhtoub S (2014) PV cell temperature/PV power output relationships homer methodology calculation. In: Conférence Internationale des Energies Renouvelables" CIER'13"/International Journal of Scientific Research & Engineering Technology. vol. 1. International Publisher &C. O. p. 0–0
Bruinenberg J, Colton L, Darmois E, Dorn J, Doyle J, Elloumi O et al (2012) CEN CENELEC ETSI Smart Grid Coordination Group on Smart Grid Reference Architecture. CEN CENELEC ETSI Technical Report. p. 98–107
Camus B, Paris T, Vaubourg J, Presse Y, Bourjot C, Ciarletta L et al (2016) MECSYCO: a Multi-agent DEVS Wrapping Platform for the Co-simulation of Complex Systems. Accessed 26 Jun 2022
Coelho VN, Cohen MW, Coelho IM, Liu N, Guimarães FG (2017) Multi-agent systems applied for energy systems integration: state-of-the-art applications and trends in microgrids. Appl Energy 187:820–832
Estebsari A, Mazzarino PR, Bottaccioli L, Patti E (2021) IoT-enabled real-time management of smart grids with demand response aggregators. IEEE Trans Ind Appl 58(1):102–112
Eugster PT, Felber PA, Guerraoui R, Kermarrec AM (2003) The many faces of publish/subscribe. ACM Comput Surveys (CSUR) 35(2):114–131
Garau M, Ghiani E, Celli G, Pilo F, Corti S (2018) Co-simulation of smart distribution network fault management and reconfiguration with lte communication. Energies 11(6):1332
Georg H, Müller SC, Dorsch N, Rehtanz C, Wietfeld C (2013) INSPIRE: integrated co-simulation of power and ICT systems for real-time evaluation. In: 2013 IEEE International Conference on Smart Grid Communications (SmartGridComm); p. 576–581
Gomes C, Thule C, Broman D, Larsen PG, Vangheluwe H (2018) Co-simulation: a survey. ACM Comput Surveys (CSUR) 51(3):1–33
IEEE Standard for Modeling and Simulation (M amp;S) High Level Architecture (HLA)—Object Model Template (OMT) Specification. IEEE Std 15162-2010 (Revision of IEEE Std 15162-2000). 2010;p. 1–110
IEEE Standard for Modeling and Simulation (M amp;S) High Level Architecture (HLA)—Object Model Template (OMT) Specification—Redline. IEEE Std 15162-2010 (Revision of IEEE Std 15162-2000) - Redline. (2010);p. 1–112
Jung T, Shah P, Weyrich M (2018) Dynamic co-simulation of internet-of-things-components using a multi-agent-system. Procedia CIRP 72:874–879
Massano M, Macii E, Patti E, Acquaviva A, Bottaccioli L (2019) A grey-box model based on unscented Kalman filter to estimate thermal dynamics in buildings. In: 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I &CPS Europe). IEEE. p. 1–6
Mattei M, Notton G, Cristofari C, Muselli M, Poggi P (2006) Calculation of the polycrystalline PV module temperature using a simple method of energy balance. Renewable Energy 31(4):553–567

Mazzarino PR, De Vizia C, Macii E, Patti E, Bottaccioli L (2021) An agent-based framework for smart grid balancing exploiting thermal flexibility of residential buildings. In: 2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe); p. 1–6
Mihal P, Schvarcbacher M, Rossi B, Pitner T (2022) Smart grids co-simulations: survey & research directions. Sustain Comput Inf Syst 35:100726
Motie Y, Belghache E, Nketsa A, Georgé JP (2018) Interoperability based dynamic data mediation using adaptive multi-agent systems for co-simulation. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). IEEE. p. 235–241
Nunna HK, Doolla S (2012) Multiagent-based distributed-energy-resource management for intelligent microgrids. IEEE Trans Industr Electron 60(4):1678–1687
Palensky P, Van Der Meer AA, Lopez CD, Joseph A, Pan K (2017) Cosimulation of intelligent power systems: fundamentals, software architecture, numerics, and coupling. IEEE Ind Electron Mag 11(1):34–50
Palensky P, Cvetkovic M, Gusain D, Joseph A (2021) Digital twins and their use in future power systems. Digital Twin 1(4):4
Palmintier B, Krishnamurthy D, Top P, Smith S, Daily J, Fuller J (2017) 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); p. 1–6
Pan Z, Xu Q, Chen C, Guan X (2016) NS3-MATLAB co-simulator for cyber-physical systems in smart grid. In: 2016 35th Chinese control conference (CCC). IEEE 2016:9831–9836
Paris T, Ciarletta L, Chevrier V (2017) Designing co-simulation with multi-agent tools: a case study with NetLogo. In: Multi-Agent Systems and Agreement Technologies. Springer. p. 253–267
Pipattanasomporn M, Feroze H, Rahman S (2009) Multi-agent systems in a distributed smart grid: design and implementation. In: 2009 IEEE/PES Power Systems Conference and Exposition. IEEE. 1–8
Reinbold V, Protopapadaki C, Tavella JP, Saelens D (2019) Assessing scalability of a low-voltage distribution grid co-simulation through functional mock-up interface. J Build Perform Simul. p. 1–13
Ringkjøb HK, Haugan PM, Solbrekke IM (2018) A review of modelling tools for energy and electricity systems with large shares of variable renewables. Renew Sustain Energy Rev 96:440–459
Roche R, Blunier B, Miraoui A, Hilaire V, Koukam A (2010) Multi-agent systems for grid energy management: a short review. In: IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society. IEEE. p. 3341–3346
Scherfke S (2014) aiomas Documentation. Accessed 26 Jun 2022
Schiera DS, Minuto FD, Bottaccioli L, Borchiellini R, Lanzini A (2019) Analysis of rooftop photovoltaics diffusion in energy community buildings by a novel Gis-and agent-based modeling co-simulation platform. IEEE Access 7:93404–93432
Schloegl F, Rohjans S, Lehnhoff S, Velasquez J, Steinbrink C, Palensky P (2015) Towards a classification scheme for co-simulation approaches in energy systems. In: 2015 International symposium on smart electric distribution systems and technologies (EDST). IEEE. 516–521
Schütte S, Scherfke S, Tröschel M (2011) Mosaik: a framework for modular simulation of active components in smart grids. In: 2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS). IEEE. 55–60
Schweiger G, Gomes C, Engel G, Hafner I, Schoeggl J, Posch A et al (2019) An empirical survey on co-simulation: promising standards, challenges and research needs. Simul Model Pract Theory 95:148–163
Sergi B, Pambour K (2022) An evaluation of co-simulation for modeling coupled natural gas and electricity networks. Energies 15(14):5277
Song J, Jiang S, Zhang P, Zhou J (2017) Real-time digital co-simulation method of smart grid for integrating large-scale demand response resources. CIRED-Open Access Proc J 2017(1):1949–1953
Steinbrink C, van der Meer AA, Cvetkovic M, Babazadeh D, Rohjans S, Palensky P et al (2018) Smart grid co-simulation with MOSAIK and HLA: a comparison study. Comput Sci-Res Dev 33(1):135–143
Steinbrink C, Blank-Babazadeh M, El-Ama A, Holly S, Lüers B, Nebel-Wenner M et al (2019) CPES testing with Mosaik: co-simulation planning, execution and analysis. Appl Sci 9(5):923
Thurner L, Scheidler A, Schäfer F, Menke J, Dollichon J, Meier F et al (2018) pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems. IEEE Trans Power Syst 33(6):6510–6521
United Nations (2022) Energy, UN-Habitat. Accessed 26 Jun 2022. Available from: https://unhabitat.org/urban-themes/energy/
Widl E, Wild C, Heussen K, Rikos E, Hoang TT (2022) Comparison of two approaches for modeling the thermal domain of multi-energy networks. In: 2022 Open Source Modelling and Simulation of Energy Systems (OSMSES). IEEE. 1–6
Zhang J, Daily J, Mast RA, Palmintier B, Krishnamurthy D, Elgindy T et al (2020) 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). p. 1–5

Publisher's Note