POLITECNICO DI TORINO

Master of Science's Degree in ICT for Smart Societies

Master of Science's Degree Thesis

COMPARISON BETWEEN TWO CO-SIMULATION FRAMEWORKS, MOSAIK AND HELICS

Supervisors

Prof. LORENZO BOTTACCIOLI

Prof. CLAUDIA DE VIZIA

Prof. EDOARDO PATTI

Candidate

JUAN GILBERTO RUEDA VÁSQUEZ

December 2022

Abstract

The Energy systems combine physical domains related directly to the processes of generation, storage, distribution, and consumption, in addition to communication technologies and software infrastructure for data and control purposes. The complexity of these heterogeneous systems makes it difficult to study them since tools of different domains must interact. In recent years, emerge a new enabling technique where global simulation of a complex system can be achieved by composing the simulations of its parts called Co-simulation.

This research has the purpose of making a comparison between two of these co-simulation frameworks, MOSAIK and HELICS. To this aim, the comparison is divided into two components theoretical and performance. The first component analyze the tools considering their conceptual architectures, giving particular importance to how each framework handles the time synchronization and the data exchange between all the co-simulation simulators. The second one center the study on the performance presented by the platforms HELICS and MOSAIK with two chosen case studies. To allow the configuration and set-up of each case study can be done in the same way, it uses a flexible system that brings a plug-and-play integration of models, simulators, and scenarios, independently of the framework. In this system, one or more models can be easily replaced without affecting the whole simulation engine, and it is possible to choose the framework you want to execute.

Each study case is composed of different simulators that are combined in a shared simulation environment. Case study one represents a simple electrical network composed of four Python models used to simulate the grid, some photovoltaic panels, and buildings. Case study two models a greater electrical system where the performance of the building is simulated with Energyplus, the heat pump with its control strategy is modeled in Modelica, household occupancy, electrical loads, photovoltaic production, smart meters, weather, and grid employ Python simulators.

Both frameworks simulate the case studies with a set of predefined scalability scenarios, i.e., each scenario run has more than one replica for one of its simulators. During these tests, data on time spent in the simulation and computational resources required for each case, each scenario, and each framework were obtained and stored. Finally, these results are presented as well as the analysis of the similarities and differences between MOSAIK and HELICS by performing the co-simulations.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Energy systems are becoming very complex over the last few years. Integrating different domains such as physical, software, and network components has led to design, operation, control, analysis, and maintenance challenges.

The buildup of state-of-the-art modeling and simulation tools that capture this interdisciplinary (cyber and physical domains) of current energy systems has become a field of research of increasing interest.

Co-simulation is one of the emerging approaches for this type of complex system. Co-simulation enables to global creation scenario with a set of diverse coupled simulators, where each one is a black box mock-up of a constituent system developed and provided by the team that is responsible for that system.

The name of co-simulation frameworks is given since one of the components is a middleware that is responsible for data exchange and temporal synchronization of all the models.

MOSAIK and HELICS are two of these called co-simulations frameworks.MOSAIK has been developed at the Oldenburg Institute for Information Technology and HELICS at the U.S. Department of Energy. This thesis has the purpose of making a comparison between these two co-simulation frameworks. The comparison is made in two sections, the particular architectural concepts of the tools, as well as results from two simulation studies implemented. The conceptual comparison gives particular importance to how each framework handles the time synchronization and the data exchange between all the co-simulation simulators. The implementing part centers the study on the performance presented by the platforms HELICS and MOSAIK with two chosen case studies.

The remainder of this thesis is structured as follows: Charter. 2 gives an explanation of the research methodology and each of its phases. Chapter. 3 presents

the literature review conducted. The theoretical description of MOSAIK and HELICS is made in Chapter. 4, followed by the conceptual comparison between both frameworks conducted in Chapter. 5. Chapter 6. describes the two case studies implemented with the respective analysis of results, and Chapter 6, finally concludes the thesis.

1.1 Problem formulation

1.2 Research Objectives

1.2.1 General Objective

Make a comparison between MOSAIK and HELICS co-simulation environments for smart city scenarios.

1.2.2 Specific Objectives

- Identify the conceptual differences and similarities between the co-simulation frameworks MOSAIK and HELICS.
- Evaluate the performance of each co-simulation framework with two case studies.
- Propose a co-simulation systems that allows to use both framework MOSAIK and HELICS from a common setup document.

Chapter 2

Research Methodology

The methodology followed during the development of the present research was structured in four main phases:

- Phase 1: State of the art: Co-simulation.
- Phase 2: Theoretical overview of the co-simulation frameworks: MOSAIK and HELICS.
- Phase 3: Conceptual comparison between MOSAIK and HELICS.
- Phase 4: Co-Simulation case studies.

Phase 1: State of the art: Co-simulation: A systematic literature review was conducted to gain knowledge of the emerging enabling technique called Co-simulation, the state-of-the-art of the different frameworks used as well as comparisons made between them, with particular attention in the frameworks MOSAIK and HELICS. The results of this first phase are presented in Chapter 3.
Phase 2: Theorethical overview of the co-simulation frameworks: MOSAIK and HELICS: An in-depth understanding of the theoretical rationale behind the MOSAIK and HELICS co-simulation frameworks was conducted in this phase. Through the available works in the literature, was studied the structure used by each framework, the approach that each on implements, and was identified their key features. The results of this second phase are presented in chapter 4.
Phase 3: Conceptual comparison between MOSAIK and HELICS: The conceptual analysis-oriented to compare the two co-simulation frameworks MOSAIK and HELICS was developed in this specific phase. The comparison was made regarding their architectures and main characteristics as the data exchange and time management. The description of the differences and similarities between both frameworks is presented in more detail in Chapter 5.
Phase 4: Co-Simulation case studies: Finally, the last phase corresponds to

comparing the results given by both frameworks from the simulation of representative tests cases. To this end, it was implemented two case studies. A detailed analysis was later carried, considering all the characteristics, advantages, and disadvantages showed by MOSAIK and HELICS over each scenario. The description of the case study and the analysis of the results are given in Chapter 6.

Chapter 3

State of the art

Nowadays, the integration of renewable energies into the electric power grid system have become widely diffused. These new technologies add a further degree of complexity transforming the whole energy system. The traditional network developed based on a centric approach with unidirectional power flow, and hierarchical topologies cannot deal with the challenges and complexity of a more distributed and flatter grid. The Smart Grid concept appears to replace the previous approach. It requires integrating Information and communication technologies (ICT), power electronics, and business applications [1], which lead to understanding the Smart Grid as a cyber physical energy system (CPES). Aiming to solve the challenges, industry and researchers jointed efforts developing components and strategies with the purpose to reach efficiency, sustainability, reliability, security, and stability in the energy grid [2].

Meanwhile, the integration of cyber-physical systems into the energy grid increases, evaluate new technological developments to guarantee functionality during the operation is getting quite complex due to the interdisciplinarity face in the network [3]. Software simulations proves to be valuable as a test stage, as well as an easily scalable test environment allowing consideration of larger-scale scenarios without stressing the actual physical infrastructure [4] and [5]. Additionally, to assess the performance of possible solutions, simulation tools offer a cost-effective approach. An overview of the tools applicable in Smart Grid research is shown in the work of Mets et al. [6].The authors identified two main groups containing most of simulations tools: the power systems that typically adopt a continuous-time model and communications network simulators that adopt a discrete event simulation approach. However, each energy system component count with a large set of simulation applications.

Given the high variety of components belonging to Smart Grids and the increased linked between elements of diverse nature, it is necessary to use environments that

allow different combinations among them [7]. This approach is known as co-simulation or coupled simulation. An integral simulation platform aims to model multidomain systems connecting the simulations of its parts to solve within their native environments [8]. In the last years, different solutions have been developed in this concern [9]. For instance, in the work by Palensky et al. [10], a prototype platform simulation tests the dynamic interaction of a flexible-demand EV charging management system. Combining a heterogeneous set of simulation tools regarding authentic user behavior, realistic battery model, and reliable electric distribution grid calculations, showed modeling capabilities, scalability, and modularity provided by this flexible environment.

In the work by Kelley et al.[11], a middleware component called FSKIT was developed to support large-scale integrated transmission and communications systems. FSKIT was used to couple a power transmission simulator with dynamic capabilities GridDyn and the open-source network simulator NS-3 obtaining high time accuracy.

How to coordinate individual simulators, numerical aspects, and software interfaces for both power and ICT domain are discussed by López et al. [12]. Together, the simulators allow researchers to analyze complex interactions and dynamics in more detail. The work by Palensky et al. [13] presents the properties of three smart grid co-simulation tools: Powerfactory, Open Modelica, and OMNET++, belonging to simulating intelligent power systems. As conclusion, the authors highlight the need for a unique language, documentation, distributed computing, validation, complexity, multigranular models, and heterogeneous models as relevant future research directions inside power systems.

Researchers increasingly link different simulators forming novel co-simulations. As a consequence, it has been a need to make classifications to have a guideline for futures works. Studies addressed in this way were found in the literature. For instance, [14], and [15] present a survey and lessons learned on the topic of CPES testbeds. Regarding general classifications among the whole set of available and used co-simulation frameworks, related information is described in [16], [5], and [3]. They offer classifications between the co-simulation tools, considering both theoretical differences and similarities.

One of the work environments developed by research teams focused on co-simulation is MOSAIK. This framework offers a composite simulation environment where models with varying properties can yield sensible and reliable results. MOSAIK is composed of four layers bridging the gap between control strategies, scenario specification, and simulation models by introducing semantic information about these [17]. MOSAIK is an attempt toward a Smart Grid specific standard for co-simulation. A detailed description of the generic interface called SimAPI and the semantic layer, additionally to a first simulation use case, is presented in [18].

The work by Rohjans et al. [19] shows the integration of a real-time power

simulator called Aristo with an implemented Multi-Agent-System (MAS) into MOSAIK. Using load and wind generation profiles is analyze the impact of control strategies with agent-based decision-making on the voltage and power flow calculations. In this case study, Mosaik connects the different agents and performs the coordination mechanism between them. An urban energy modeling system is developed by Wang et al. [20]. Its modularity design allows integration of different urban energy simulation tools encapsulated in Functional Mockup Units (FMU). A coupled simulation environment compose by EnergyPlus simulating a building's energy flows. Nottingham Multi-Agent Stochastic Simulation (No-MASS) generates synthetic populations of buildings' occupants and their energy-related behaviors. And Mosaik as master orchestrator managing the different simulation scenarios and data flow between the individual components, evaluated two study cases: a show box office and multiple buildings.

An extended CPES test environment is presented in [4]. It uses a virtual power plant with three wind turbines and two industrial loads located in a grid with some distributed households and industrial load. Additionally, units forecast the upcoming power consumption and the tool Pandapower to calculate power flow in the grid. Using the framework MOSAIK the whole scenario is implemented, looking to perform congestion management to prevent transformer overloads. In conclusion, MOSAIK shows to be a flexible and usable solution for CPES testing across multiple domains. The centralized scheduling concept of MOSAIK allows treating all data exchanged identically without additional configuration. Other frameworks, especially those based on HLA, require the user to provide more specification in the interaction between the simulators.

The work by Barbierato et al. [21] proposes a multi-model co- simulation platform designed for various general-purpose services for smart gird management following an event-driven approach. With Functional Mock-up Interface (FMI), the simulation models have been extended. Using MOSAIK as an orchestrator, the integrated simulators are EnergyPlus for building model, Modelica for Heat Pump system, photovoltaic system, household electricity behavior model, and weather model. The platform is tested in a hypothetical and realistic house located at Turin.

A co-simulation approach using MOSAIK applied on a power system with grid-forming converters is presented by Farrokhseresht et al. [22]. The power system modeled in Powerfactory and the controller of the converter-based generator modeled in Simulink are coupled. The study tests the control's functionality to validate their efficacy in a co-simulation setting against a monolithic Powerfactory simulation and applies it for transient stability analysis.

A new recently co-simulation environment developed is the Hierarchical Engine for Large-scale Infrastructure co-simulation (HELICS). This framework is built on the collective experience of multiple national laboratories of Unite State of America.

HELICS allows leveraging existing off-the-shelf tools for transmission, distribution, communication, and distributed market. Its layered architecture enables both high-performance simulations and efficient software development [23]. HELICS was developed by the same team that in previous years design the framework called FNCS, as well as FNCS utilizes a federated approach HLICS does. It can be considered that HELICS is the evolution of FNCS which was discontinued in 2015 [24]. It was found that the framework FNCS is used in research-oriented to analyze scenarios of cybersecurity attacks and evaluate their consequences [25] and [26]. These studies integrate the simulation tools GridLAB-D and NS-3 through the FNCS environment. Since the simulation tools mentioned can be coupled as well on HELICS, this suggests that the framework could employ in the same study field. The integration of the Network Simulator 3 (NS-3) into HELICS is described in the work by Zhang et al. [27]. The authors analyzed the impact of hybrid communications design on the Distributed Energy Resources (DER), monitoring network performance metrics of latency and packet loss. The used case study was a DER grid composed of 51 PV nodes, 275 smart meters, 10 data concentrators, and one edge router, divided into ten neighborhood areas.

Bharati and Ajjarapu [28] presents a developed Transmission and Distribution (TD) co-simulation combining the commercial transmission system solvers PSS/E and the accurate distribution system solver GridLAB-D using HELICS. The study shows the high-inertia and a low-inertia induction motor response in the distribution system to a fault in the transmission system.

An implemented market co-simulation in HELICS, analyzing two DER penetrations levels, is described in [29]. Modeling the interactions between a House model, a House controller, an Energy orders broker, a Grid simulator, and the Market solver, the economics results are compared to full retail net energy metering. It was demonstrated the potential for distribution markets to enter into the discussion regarding post-net metering paradigms.

Chapter 4

Theoretical overview

4.1 MOSAIK

The Smart Grid domain has comprehensive technologies where their specific expertise must be adequately integrated to provide a sustainable and reliable power supply. The MOSAIK framework, as a co-simulation approach, provides a flexible architecture enabling the integration of these existing models and platforms into large-scale simulation scenarios[17].

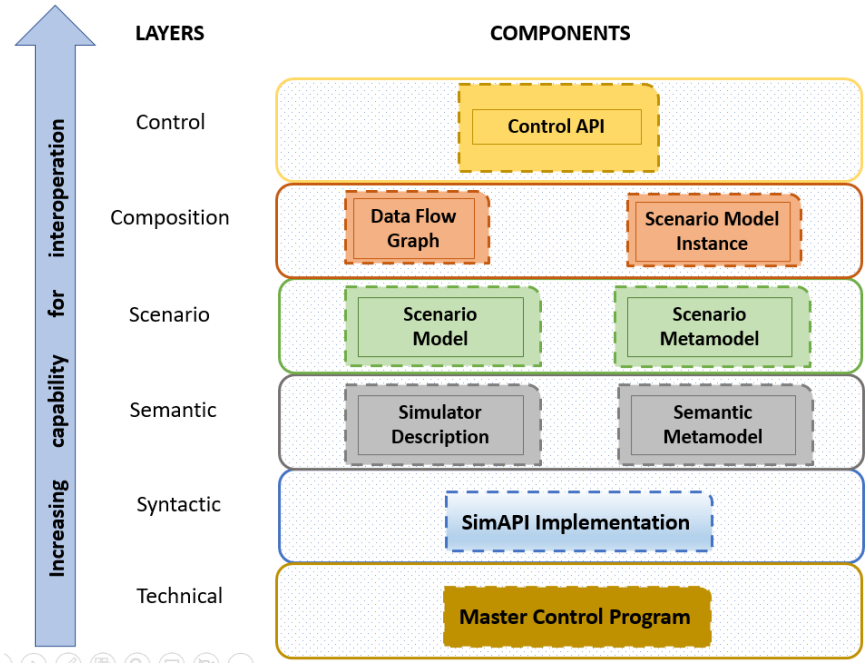

Figure 4.1: Mosaik conceptual architecture

The MOSAIK conceptual architecture is composed of six layers, as is shown in figure 4.1, which allows for addressing goals of syntactic semantic interoperability, as well as scenario modeling and control strategy integration. This architecture was developed based on two computer science models: Levels of Conceptual Interoperability (LCIM) and the Architecture for Modeling and Simulation [30].

Each layer fulfills a specific function as well as meets a set of requirements. To get a better understanding of each one, they will be explained below.

Technical layer: It consists of computational and network hardware. It manages the simulator processes, including initialization, monitoring, and stopping as required. Additionally, it addresses the distributed simulation infrastructure meeting the requirement of running on multiple servers.

Syntactic layer: It enables simulators to interact with their models through a well-defined interface called SimAPI, which must be implemented by each. As main features of this layer, it can be highlighted:

- It can retrieve relations between the entities of its simulation models.
- It allows the provision of information about the static properties of the entities.
- It supports the transmission of complex data types.
- It enables to have variable entity quantities as well as variable-specific inputs and outputs.
- It allows the integration of commercial off-the-shelf simulators.
- The SimApi uses a fixed time step to handle the simulator's process. This allows integrating simulators with different time paradigms as continuous and discrete models.
- It enables heterogeneous simulations where simulators implemented with different languages, tools, and frameworks can be integrated.

Semantic layer: It allows the creation of a reference data model giving a common understanding of the exchanged data between simulators. The Semantic layer employs a metamodel called Semantic metamodel, where is placed the information obtained about the simulation models, their entities, and information flows. Other relevant aspects of this layer are:

- It gives access about the static data of each entity /model.
- It offers means to handle entities with dynamic data.

- It addresses the definition of the structure of models with complex data types.
- It provides information regarding the number of entity instances each model contains.
- It provides in the simulator description what is the step size of each simulator.
- It gives information about the data flows and their connections.
- It offers the mechanism to define the required parameter values of each simulator.

Scenario layer: It provides the formal description of Smart grid scenarios with a metamodel named scenario metamodel, where are given the reference elements describing each simulator. The scenario layer gives responds to the following requirements:

- It provides the scenario specification mechanism that enables the automatic composition of simulation models.
- It allows the connection of entities from entity sets, and it defines the rules to connect and reuse them, enabling the possibility of creating large scenarios.
- It allows the composition of entities based on their attributes.
- It integrates large numbers of different entities in an automated fashion.
- It allows for defining the step size of simulators.
- It enables the specification of the parameter value of simulators.

Composition layer: This layer translates the work of each of the layers mentioned above into the following set of tasks to be managed.

- It interprets the scenario metamodel.
- It initializes the simulators and models.
- It establishes the data flows between the simulated entities.
- It is in charge of advancing the simulators in the correct order.

Control layer: It aims to access and control the state of simulated entities using the interface called ControlAPI. It meets the above requirements:

- It provides functionalities to get information regarding the physical components.

- It gives access to whole model entities relations.
- It gets information about the static entities' attributes.
- It provides a mechanism for synchronizing control strategies.

4.2 HELICS

Energy systems face a growing of distributed resources into the grid and their increasingly intertwined with communication (ICT) systems. This interdependency has prompted the development of advanced modeling and simulation tools that capture both cyber and physical domains.

The co-simulation enables the global simulation of a coupled system composed of different simulators. This technique allows a better understanding of each energy system component and test the reliability, efficiency, and cost-effectiveness.

HELICS (Hierarchical Engine for Large-scale Infrastructure Co-Simulation) is a new, open-source co-simulation platform that has been developed by the U.S. Department of Energy in partnership with multiple national laboratories.

HELICS offers a modular, high-performance, scalable, cross-platform co-simulation framework for modeling cyber-physical-energy systems. It can support large-scale (10,000,000+ federates) co-simulations with off-the-shelf power-system, communication, market, and end-user tools; furthermore, both event-driven and time series simulation.

The development of HELICS gives response to the following requirements:

- It supports a various range of co-simulation from small-scale interaction to large-scale interconnection.
- It supports all operating systems.
- It enables commercial tools.
- It allows the easy build-up of wide co-simulation scenarios.
- It enables the integration of diverse, existing simulation tools through an easy interface. Open-source.
- It supports discrete-event simulation, quasi-steady-state time series, and phasor dynamics.
- It enables inter-federate convergence.

- It ensures transmission-distribution power flow convergence.

HELICS provides a rich set of APIs written in Python, C, Java, and Matlab. Hence, multiple simulation models "federates" from various domains can interact with high performance and create a larger co-simulation "federation" able to capture rich interactions.

HELICS employs a layered architecture that enables clean and modular maintainability, besides a parallel development of each layer which is possible through APIs between them, allowing each layer to make internal changes and optimizations without impacting the others.

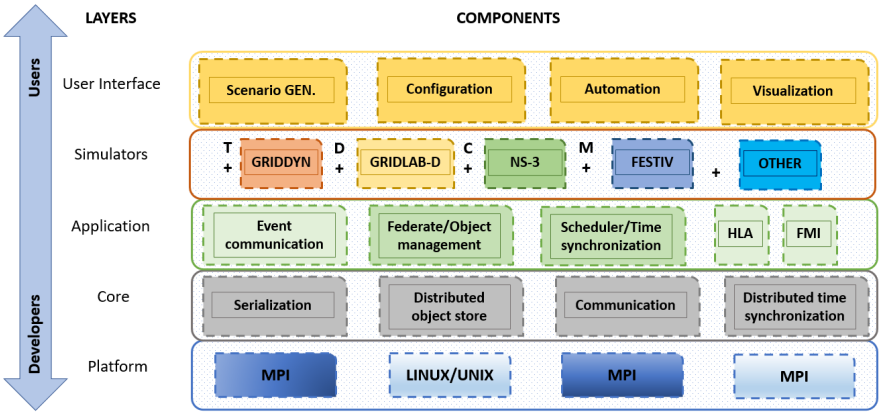

Figure 4.2: Helics conceptual architecture

It consists of the following five layers:

Platform layer: The software associated with this layer is written in cross-platform C++, using C++14 features. It allows the integration of existing packages and the use of other co-simulation tools. The platform layer ensures HELICS can work across multiple operating systems and multiple computational scales. To achieve this goal, it employs two communications interfaces, Message Passing Interfaces (MPI) and ZeroMQ, that will be used depending on the system latency [23][27].

Core layer: The Core layer works like an interface that is supported by either ZeroMQ or MPI-based backend. It manages two essential mechanisms, the data exchange and time synchronization of both types of federates: the discrete event and the time series. It provides the different constructs that allow modeling any interaction between federates. The HELICS federates can register endpoints, with which it is possible to do special operations like:

- Direct pairwise communication.
- Communication latency.
- Complex message interactions.

Therefore, the core layer enables HELICS to work value-based, and message-based interactions and, joined with the application layer, facilitates model network communications.

Finally, the core layer is where federates register to the federation, and where their time management is coordinated. HELICS allows to federates to work with different time scales and co-iterate at any time step[23][27].

Application layer: The Application layer is a low-level interface that supports applications federates interacting with the co-simulation framework and the Core API, making it easier for generic applications of different types of federates (Value, Message, Message filter, and Programming interface) to interact in a flexible fashion.

As a low-level interface, this layer enables HELICS to support other low-level interfaces, such as High-Level Architecture (HLA) and Functional Mockup Interface(FMI)[23][27].

Simulator layer: The Simulator layer provides two key extensions: standardized data exchange patterns and higher-level API[23][27]. These extensions allow HELICS to support a variety of off-of-shelt simulators, such as:

- Transmission simulators (e.g., GridDyn, PSS/E).
- Distribution simulators (e.g., GridLAB-D, OpenDSS, CYMDIST).
- Communication simulators (e.g., NS3).
- Market simulators (e.g., FESTIV).
- Customized controllers.

User Interface layer: Thanks to the User Interface Layer, co-simulations at any scale have shown to be easy integration to integrate into HELICS. This layer provides tools that allow:

- Manage and convert input data,
- Generate scenarios and populate required data,
- Automate simulations execution, and

- Parse results,

with and standardized approach. [23][27].

Chapter 5

Conceptual Comparison

The above Chapter showed the devised in their conceptual architecture of the co-simulation framework MOSAIK and HELICS. Given a quick and overall overview, both frameworks show similarities and differences. Thus, this Chapter gives a deeper theoretical comparison between MOSAIK and HELICS, considering data exchange and time management as the relevant topics to analyze.

In both setups, the main goal is to use independently existing subsystems called simulator in MOSAIK or federate in HELICS in a shared context to execute a coordinated simulation of a given Smart Grid scenario. Therefore, co-simulation frameworks must synchronize the processes of each subsystem and manage the exchange of data between them. A co-simulation framework must provide the following aspects [9]:

- Communication between simulators/federates and framework.
- Handlers for different kinds of processes.
- To allow using simulators/federates of different natures.
- To manage data-flow and step-wise execution.

Both frameworks develop the above-presented aspects, given a response in their way.

As is shown in figure 5.1, MOSAIK presents four components: The Interface (MOSAIK Sim-API), the Scenario-API, the SIM-Manager and the Scheduler.

The Interface (Mosaik Sim-API) defines the syntactic integration between the available simulator and the framework. This component aims to capture semantics for the simulators and their models, which is the basis for automatic composition. It should be implemented for each simulator allowing to map the internal simulator paradigm to a discrete-time approach.

The Scenario-API provides the means for the creation of co-simulation scenarios. Making use of different commands is possible to start simulators, instantiate models from them, and connects them with a pure Python interface that, through a semantic meta-model, enables the definition of the physical topology[9]. Through this script, Mosaik provides the means by which the user can:

- Specify the simulators and the model entities.
- To set initial events.
- To parameterize entities and set the time period of the simulation.
- To define interconnection and cyclic data-flows.
- To create user-defined connection rules between model entities.
- To call extra methods of a simulator.

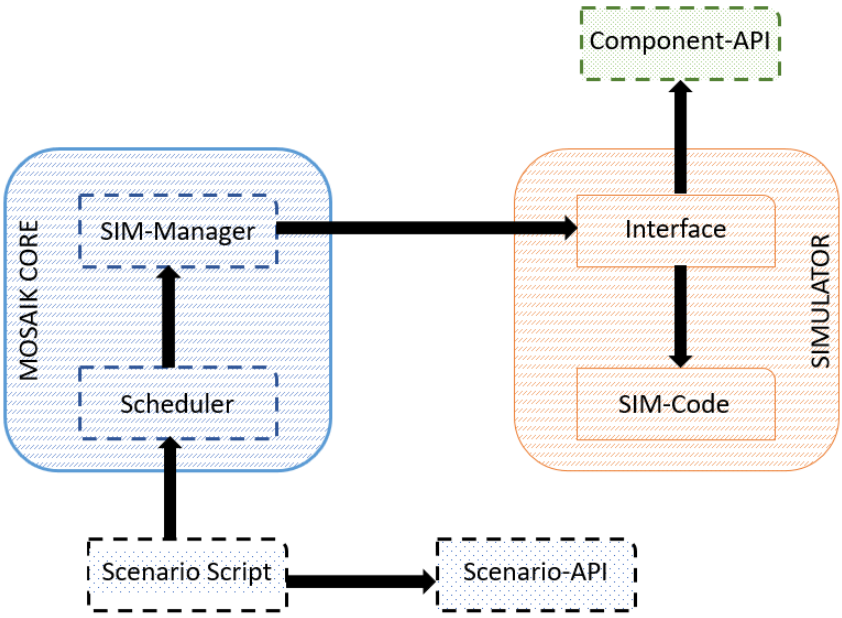

Figure 5.1: Mosaik configuration

Both missing components are part of was is called the **Mosaik software core**. In Mosaik, the data exchange is managed by a software core consisting of two components the SIM-Manager and the Scheduler. The sim manager is responsible for starting and handling the external simulator processes involved in a simulation as well as for communication with them. The Scheduler is in charge of organizing the whole flow of data exchange between the simulators; this is done based on

a common simulation clock that is established. Usually, in its current state, the scheduler uses discrete time.

Mosaik uses standard TCP sockets. When a simulator starts, it needs to provide a server socket that Mosaik can connect to. The interaction between the framework and the simulators is made using a network messages composed of:

1. Four bytes long header: It is an uint32 and stores the number of bytes in the payload.

2. Payload of arbitrary length: It is an UTF-8 encoded JSON list containing the message type, a message ID and the actual content.

Every request sent by a party must be responded to by the other party since the framework uses the request-reply pattern. The type of messages is 0 for request, 1 for reply with success, and 2 for reply with failure. The message ID is an integer that is unique for every request that a network socket makes.

With information supplied by the user in the two previous components (Scenario-API and SIM-API), MOSAIK knows which simulators to use, which models it can instantiate, as well as the parameters and attributes that each one has. All this allows the SIM-manager to guarantee the correct flow of information. Usually, the attributes of the models handle units and different types of data; it is important to highlight that MOSAIK does not take this into account; therefore, it is the user who must be careful to verify these factors when making connections between simulators.

Finally, the MOSAIK software core making use of the interface calls init, create, step, and get data is ready to interact with every entity of the simulation scenario.

Once the scenario has been defined and initiated, the Scheduler becomes active. It aims to ensure that each simulator is running in a synchronized fashion so that it can properly control the information flow in the co-simulation.

Coordinating the simulator executions to advance the simulation time is the most essential task of the scheduler in MOSAIK.

At the beginning of a simulation, all simulators are at time 0. The mechanism employed by MOSAIK makes the scheduler monitor the time step of each simulator in order to manage its execution. The scheduler will be in charge of informing the simulator of the time in which it must enter in execution, and in turn, the simulator must inform the scheduler of the next time step once the current one is finished. The dynamics used by MOSAIK to perform the time step make it possible to have simulators with fixed and variable steps, always bearing in mind that a simulator can only enter execution if and only if it has received the inputs correctly.

Based on the Scenario-API, MOSAIK knows how is the information flow between all simulators. Therefore, employing the API call *get_data()* it requests from each

simulator, the information that the other simulators need, together with the time tag in which it was obtained, , at the instant the simulator ends stepping.

In this way, the scheduler stores the information concerning all the *inputs* of the co-simulation components at each time step. Additionally, the scheduler must provide the respective information at the time when each simulator makes the *step()* API call.

Therefore, it should be noted that in MOSAIK, the storage and management of all the data flowing in the co-simulation are performed by the scheduler and the sim manager so that the simulators do not really interact with each other at any time, and the framework always plays the role of an intermediary. At the beginning of the co-simulation, the maximum simulation time (world.run(until=END)) must be set. Each simulator enters into execution according to the step time it has communicated and when other simulators require its data. In the case that the step time communicated by the simulator is greater than the until time, its work will be finished.

In resume, a simulator that does not supply data to the other simulators will perform its processes until it meets the condition $t_{next} > t_{until}$. And a simulator that does supply inputs will be active until the other components of the simulation have stopped.

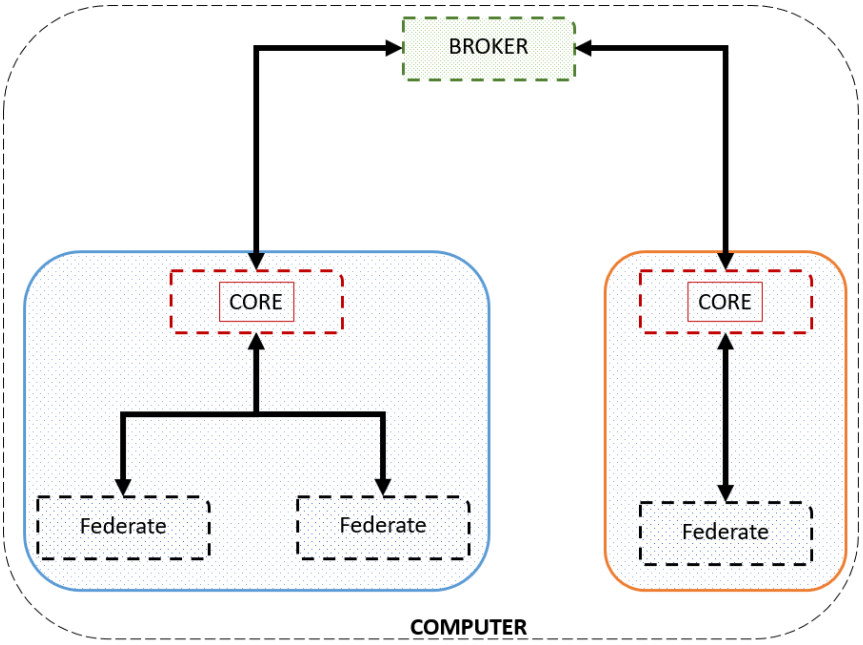

Figure 5.2: Helics configuration

In figure 5.2 is shown the most common configuration in HELICS. First of all,

it is important to clarify the meaning of three key terms in the HELICS environment.

Federate: This is the name given to instances of simulators that are already running. In a co-simulation scenario, the set of federates is called a federation.

Core: It is the software provided by the environment and allows the simulator to become a federate. Normally each federate has a core, but there may be cases where different federates share a core, as depicted in Figure 5.2. The HELICS environment offers a set of different core technologies for the user to choose according to his needs. The type of cores are the following:

- MPI: The message-passing interface is employed in a High performance computing cluster.
- IPC: It is called the interprocess core. It is used when we are simulating with a single compute node. The key features are that it uses memory-mapped files to transfer data and leverages Boots' interprocess communication. It can not be used with multi-tiered brokers.
- UDP: It is primary use with highly reliable networking. It employs IP messages without delivery guaranteed.
- TCP:It is an alternative to ZMQ when this core type is not available.
- ZMQ: It is the default core type. It uses the REQ/REP or PUSH/PULL mechanics for priority and non-priority communications, respectively. It provides a robust interaction in federation with multiple compute nodes.

Broker: The broker is a key component of HELICS. It is an executable, which allows maintaining the synchronization of the federates and managing the exchange of messages, as each data sent by a federate is received by the broker and then delivered to the destination federate.

Going into detail, HELICS enables to define federates according to the nature of the messages they are passing to and from the federation:

1. Value federates:Value federates are the best option to model physics of a system. It interacts with the federation using a publish-and-subscribe mechanism. This type of federate allows working with different types of data, such as floats, numbers, integers, strings, complex numbers, and arrays. It provides support for verifying matching between these as well as unit conversions. Additionally, it provides several functions that allow one to know if a value has been updated, the time of this event, and retrieve it. The federated value allows HELICS to perform co-simulations with FMUs, as it has features similar to those of a co-simulated FMU.

2. Message federates: The message federates aim to simulate ICT models. Usually, simulators modeling control signals and measurements suit this message federate. One of the main characteristics is that this type of federate must specify the source, destination, and time of the signal transmitted. Message federates use endpoint interfaces that allow them to interact with the federation. The endpoints act as an address to send and receive data. It is possible to define filters in these federates, but there has to be associated with each endpoint.

3. Combination federates: In HELICS, it is possible for any type of federate to subscribe to the publications of a valued federate. This means that we can make connections from message federates to value federates as long as the endpoint is configured with the respective publication. This type of co-simulation configuration is performed using the combined federates. It is important to highlight that in HELICS, filters can be applied to messages but not to values.

Each federate requires a configuration, which must be done by the user. There are two options, a JSON file or directly in the code with the API calls available in HELICS. In general, a basic configuration must have the following parameters defined: the name of the federate, which must be unique; the type of core, the subscriptions and publications, endpoints, and the time step size.

The time synchronization in HELICS is handled by each Federate and Core through different API calls. It is the job of every federate to determine its own work time and make a request time. This HELICS function blocks the execution of the federate thread, making the federate wait for an answer within the granted time, allowing it to continue the execution. During this waiting time, the federate has nothing to do until the next requesting time. For its part, once a time request is received, the HELICS core has two values to grant: the requested time (failing that the next available time) or an early valid time, which represents a particular case because this would imply awakening the federate mandatory so that it can make changes in its boundary conditions.

The framework has coordination between all the cores that are part of a federation to ensure that when a core grants time to a federate, it does not occur in the past. A HELICS co-simulation under normal conditions ends when all federates have received the maximum default time from the framework or when all federates notify the broker that they have finished. By means of the configuration JSON file, it is possible to establish some time specifications for the federates using a wide variety of timing parameters enabled by the HELICS framework. The time parameters are the following:

Period: Defines the resolution of the federate and forces time grants to specific intervals.
Time delta: The granted time has a minimun interval from the last one.
Offset: Amount of time added to the period.
Uninterruptible: The granted time will be always the requested time, even when the federate receives new values on any of its inputs.
Wait for current time update: The granted time will be allways the last one at a given time, making sure that all the other federates have produced outputs for that time.

Table 5.1: Mosaik and Helics: comparison between frameworks.

Component	Category	Mosaik	Helics
Time management	Handler	Scheduler	Individually each Federate
	Time-domain	Discrete	Discrete
	Step-size	Variable	Variable
	Step-size request	After every step	After every step, never, hybrid
	Step-size components	Request time	Request time and granted time
Data Exchange	Protocol inside framework	TCP sockets	ZMQ, UDP, TCP, MPI
	Definition	In Scenario-API	In Publications, Subscriptions, Endpoints
	Mesage type	Four bytes header plus a pay-load	Value, information packet, hybrid
	Data-type validation	Made to reception	Made to sending

Chapter 6

Co-Simulation Cases study

6.1 Cases study description

This section presents tests carried out with the frameworks (MOSAIK and HELICS), with the scope of evaluating the performance of each one in a co-simulation scenario. In line with Schiera et al. , it is proposed a co-simulation system that allows running scenarios in both frameworks from one YAML document that describes the whole composition of the case. Two case studies were used, and their YAML scenario schemes were made and analyzed the simulation's time and memory resources employed for each one during the entire co-simulation case. Finally, this information was saved and presented in the results section. Since the main objective of this thesis is to evaluate the comparison in performance between HELICS and MOSAIK under case studies, further specific analyses of each test concerning particular results of each context were not done because they are out of the goal of this study.

6.1.1 Case 1

This case study represents a small electrical system composed of four elements, which are the following: 1)The simulator is called 'Pypower'; it models an electrical network consisting of a transformer and thirty-seven nodes. 2)The simulator is called 'Householdsim'; it models the energetic behavior of a house. The simulator 'CSV'; represents a group of photovoltaic panels. 4) The last simulator is 'HDF5', a database where all the electrical calculations and scenario information are stored.

6.1.2 Case 2

The scenario consists of two encapsulated simulators in FMUs, the EnergyPlus building model and a Modelica-based Electric Heat Pump model, plus a control system model called Scheduler. Furthermore, a photovoltaic system, household

behavior, and weather data are provided to the building by standalone Python simulators, and two virtual Smart Meter models represent buses of the electrical grid and Smart meters simulators. Based on [31]

6.2 Co-simulation setup

Based on [31], it is proposed a YAML document, the tree diagram of which is shown in Figure 6.1. It contains all the needed information that defines our simulation scenario, the simulators we are going to employ, and the specific settings that each of them is going to use. Additionally, this document must also detail the system connections, i.e., the simulators that must communicate, the type of communication, and the parameters that will be exchanged.

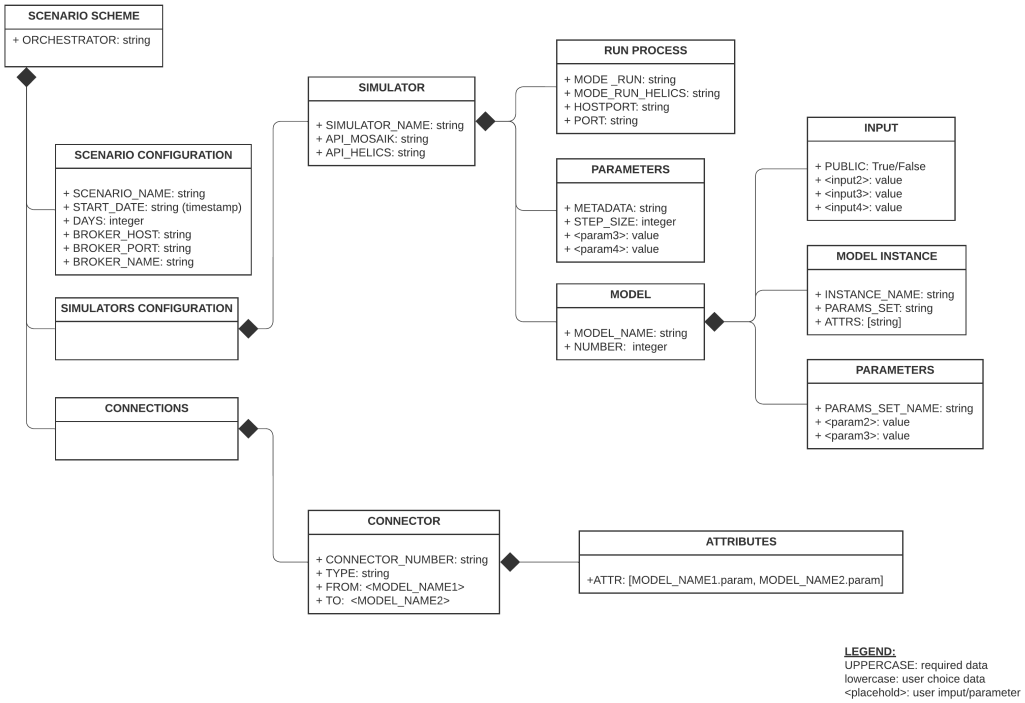

Figure 6.1: Tree diagram of Scenario YAML template

The YAML document allows, through the parameter 'ORCHESTRATOR', to choose which framework (MOSAIK or HELICS) will be used in the simulation, and in an automatic fashion, all the information consigned in the document will be configured, and the execution will start. Another highlight of this proposal is the option that is enabled to generate more than one replica of each of the simulators/federates that are being simulated and thus create scalability. This

option is enabled through the 'NUMBER' parameter that appears in each model used.

The YAML document has three main sections: The scenario Configuration, the Simulator Configuration, and the Connections.

Scenario Configuration: Here, the main information of the simulation case is detailed, such as the name, simulation duration, the framework to be used, and the IP address of the host computer.

Simulators Configuration: This second part describes in detail the different simulators that we are going to use, i.e., the simulator name, the MOSAIk and HELICS API, the IP and port of the computer used, the simulation step, parameters and variables needed to work, name of the model used, etc., must be specified.

Connections: This last component describes which simulators need to communicate, and for this purpose, it must be specified using the name of the model who sends information, who receives it, what type of communication they use, and what are the names of the variables and parameters they exchange. There may be a special case where there is more than one model with the same name, so the following must be taken into account: If all my models with the same name must perform the same communication, in the connection, I must only write the name of the general model without number and in this way the connection is established for all. If, on the contrary, there is a specific connection and a model must talk to another specific model, I must write just the name of the model with the number, so the connection is not general.

6.2.1 Case 1

Case 1 consists of a small power grid, some households, and PV systems, as is shown in figure 6.1. The Household model processes data from an external NumPy .npz file. The file contains some load profiles for a given period of time. It contains ID lists that describe which load profile belongs to which node ID in the Pypower grid. Internally, the model works with minutes and has a time resolution of 15 minutes. The CSV model also takes data from an external .csv file, given a power value with a time resolution of 1 minute for a whole year. Each created instance of CSV represents a PV panel that is associated with different nodes of the Pypower grid in a random fashion. The Pypower is a bus-branch model to represent power grids. It is composed of thirty-seven nodes/buses that are connected via branches/lines. The transformer is just a special kind of branch, and the buses are divided into three sub-types: the reference bus, PQ buses, and PU buses. For PQ buses, the (re)active power P and Q are given, and the model will calculate the voltage magnitude and angle for these nodes. PU buses provide active power and a constant voltage; thus, the model computes the reactive power for these buses. Finally, the HDF5

simulator is a relational database in which are saved all the calculations getting by the previously described three models.

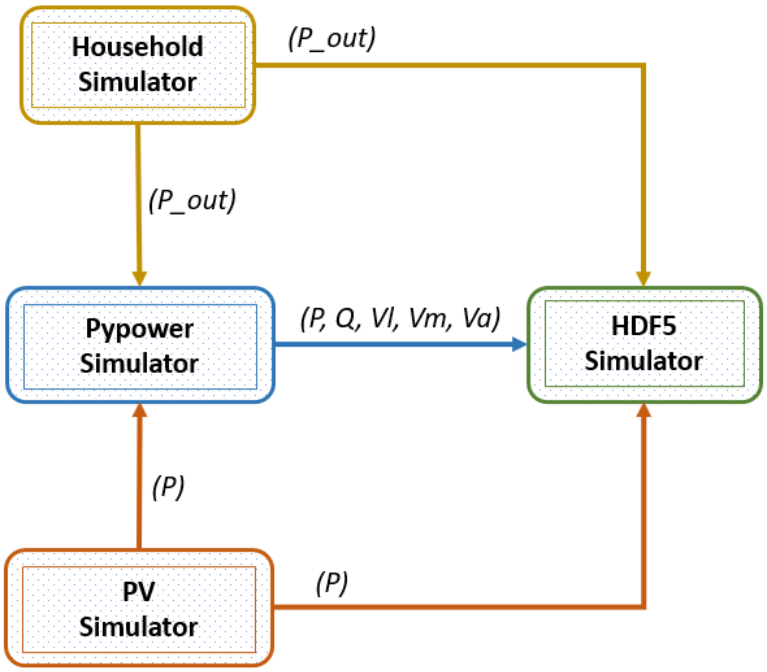

Figure 6.2: Block diagram of co-simulation scenario Case 1

6.2.2 Case 2

The scenario of Case 2 is established, as shown in figure 6.1, and consists of the following simulator blocks.

The building model uses the software EnergyPlus, which is an open-source detailed building energy modeling engine that allows performing many calculations regarding energy consumption in buildings. The potential provided by Energy Plus and its extensions, combined with the possibility of exporting the building model as an FMU, unlock a perfect integration within the co-simulation platform, allowing flexibility and composability of building models with different levels of complexity and design in function of the modelist's choices and the scenario objectives. EnergyPlus can perform simulations with a minimum time step of one minute up to one hour.

The Meteo model gives weather data at the time step required by the other models. The household model called Home represents household electricity behavior and thermal gains with a resolution of 10 minutes. The Electric Heat Pump (EHP)

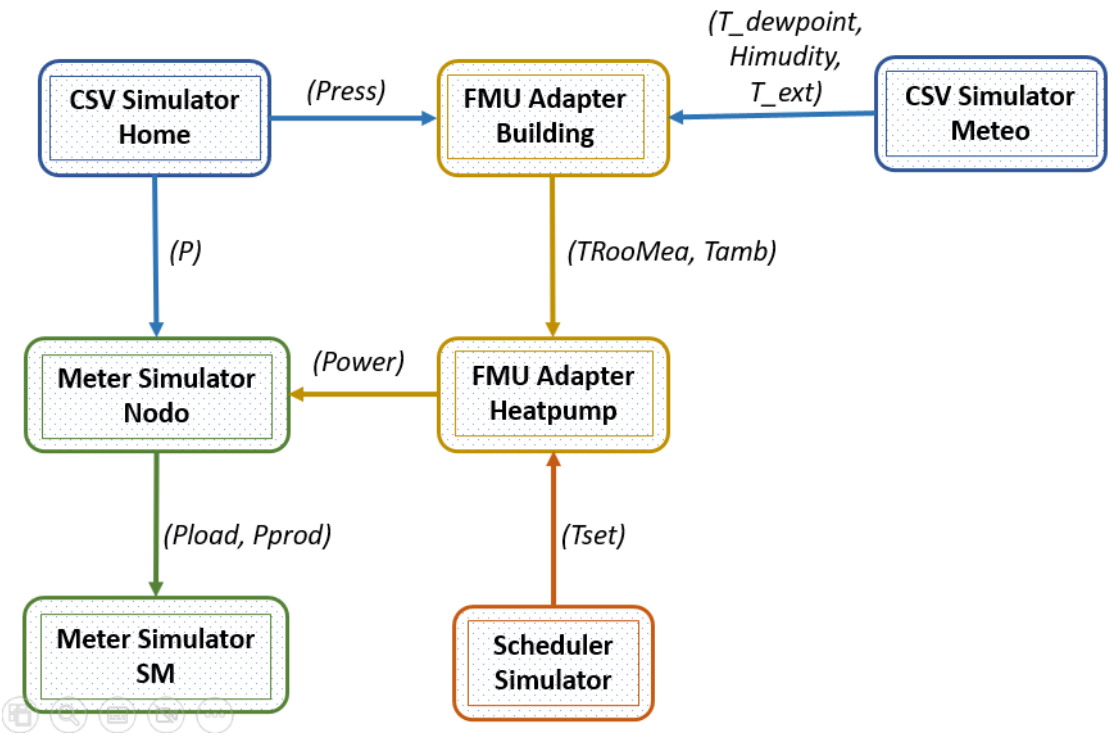

Figure 6.3: Block diagram of co-simulation scenario Case 2

simulator has been developed using the open-source OpenModelica modeling and simulation environment. This model computes the sensible heat gain required to maintain the set-point temperature Tset in rooms. The output of the Heatpum FMU is the heat requested by the building block through the heating system. There is a need to use a control system with the Heatpump simulator, and this function is done by the Scheduler model. The Scheduler goal is to maintain the desired set-point Tset by implementing a Proportional-Integral-Derivative (PID) controller that acts on the water mass flow rate of the heating system through regulation of the control valve actuator CV. Finally, the meter simulator provides the physical and data interface between the building system and the distribution network, modeling a Smart meter and the nodes/buses of the grid. The meter simulator can perform the simulation with whatever time step resolution without any limitation.

6.3 Results

To analyze how each framework behaves in terms of simulation time and required computational resources, different scenarios were defined for each case study as

shown in table 6.1 and table 6.2, where the replications of the simulators/federates that compose it vary.

The execution of the co-simulations was performed using two servers (or nodes) owned by Politecnico di Torino, interconnected through a local network. Each network node is an Intel® Xeon® Processor (Skylake, IBRS) CPU@2.294Ghz 32 Cores with 128GB RAM.

In case study 1, the simulators/federates were distributed across the two nodes as follows:

- **Node A:** Pypower Simulator and HDF5 database.
- **Node B:** Household Simulator and PV Simulator

In case study 2, the simulators/federates were distributed across the two nodes as follows:

- **Node A:** FMU Adapter Building, FMU Adapter Heatpump, Scheduler Simulator and CSV Meteo Simulator.
- **Node B:** CSV Home Simulator, Meter Nodo Simulator and Meter SM Simulator.

6.3.1 Case 1

The scenarios that were defined in this case study were:

Table 6.1: Case study 1: Defined scenarios.

Replicas of each simulator in the different scenarios					
Simulator	Scenario 1	Scenario 10	Scenario 100	Scenario 500	Scenario 1000
Household	1	10	100	500	1000
Pypower	1	1	1	1	1
CSV	1	10	100	500	1000
HDF5	1	1	1	1	1

Each scenario was executed with both frameworks MOSAIk and HELICS, gathering in every co-simulation whole the information regarding the simulation time and the computational resources required. These results are shown in Figure 6.3 and 6.4.

Figure 6.3 depicts how the simulation time employ by MOSAIK is always higher than HELICS in each scenario. It is observed that the difference between the curves appears constant as the scenarios increase, but a small tendency to increase the time difference in favor of HELICS with larger scenarios should be highlighted.

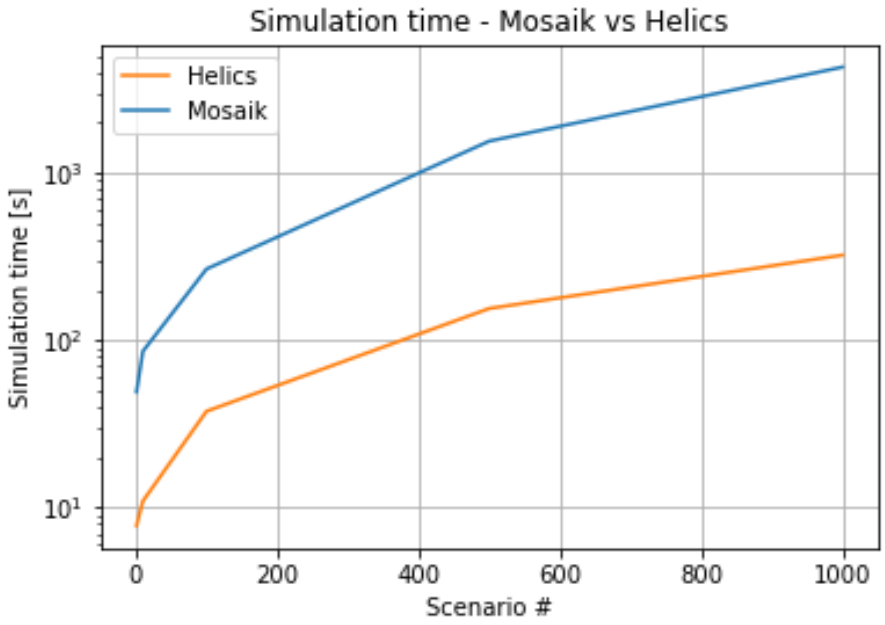

Figure 6.4: Simulation time Case 1

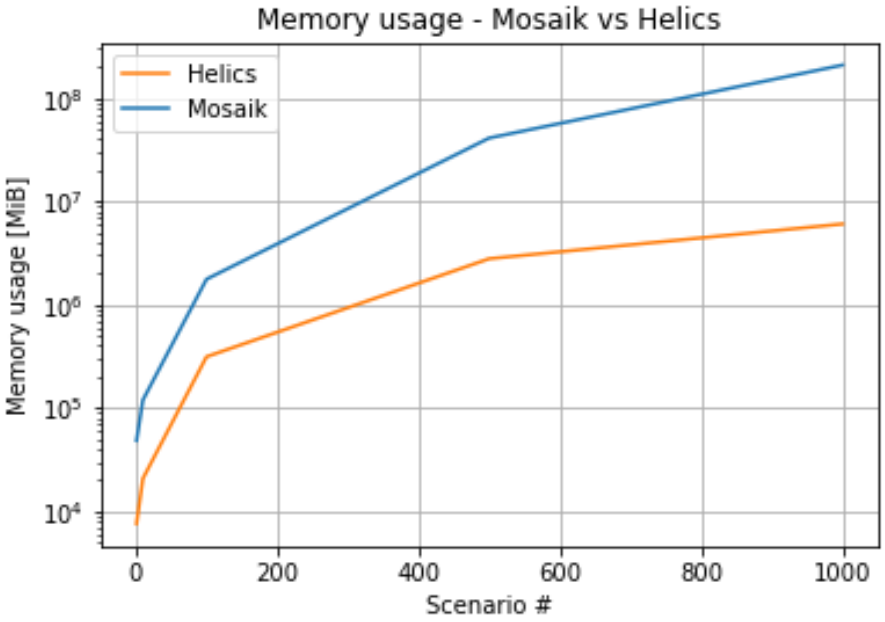

Figure 6.5: Memory usage Case 1

Figure 6.4 depicts the computational resources required by MOSAIK and HELICS with each scenario. This feature shows as well that MOSAIK is using more resources than HELICS in every co-simulations. The figure shows how in scenarios 1 and 10, there is a constant increase in the two curves. From scenario 100 onwards and as the scenarios continue to grow larger, it is clear that MOSAIK uses much more resources than HELICS and that this difference will increase with each new co-simulation.

6.3.2 Case 2

The scenarios that were defined in this case study were:

Table 6.2: Case study 2: Defined scenarios.

Replicas of each simulator in the different scenarios					
Simulator	Scenario 1	Scenario 10	Scenario 100	Scenario 500	Scenario 1000
Scheduler	1	1	1	1	1
FMU Heatpump	1	2	5	10	20
FMU Building	1	2	5	10	20
CSV Home	1	5	50	250	500
CSV Meteo	1	5	50	250	500
Meter Nodo	1	5	50	250	500
Meter SM	1	5	50	250	500

As in the above case, each scenario was executed with both frameworks MOSAIk and HELICS, and during the execution was recorded the information regarding the simulation time and the computational resources required. These results are shown in Figure 6.5 and 6.6.

In Figure 6.6 is shown the time that was spent during the simulation with MOSAIK and HELICS. This figure shows that during the different scenarios, the frameworks spent relatively close time if we compare it with what was observed in the previous case. Even though, in this case study, the difference between the curves is smaller, the tendency of HELICS to use less time than MOSAIK is maintained, and it should also be noted that as the number of simulators/federates in the co-simulation increases, the time difference starts to increase making the speed of HELICS more noticeable.

Figure 6.7 depicts the computational resources required in the simulation with MOSAIK and HELICS. In general terms, it can be said that the result is that MOSAIK is using more resources than HELICS in every co-simulation.

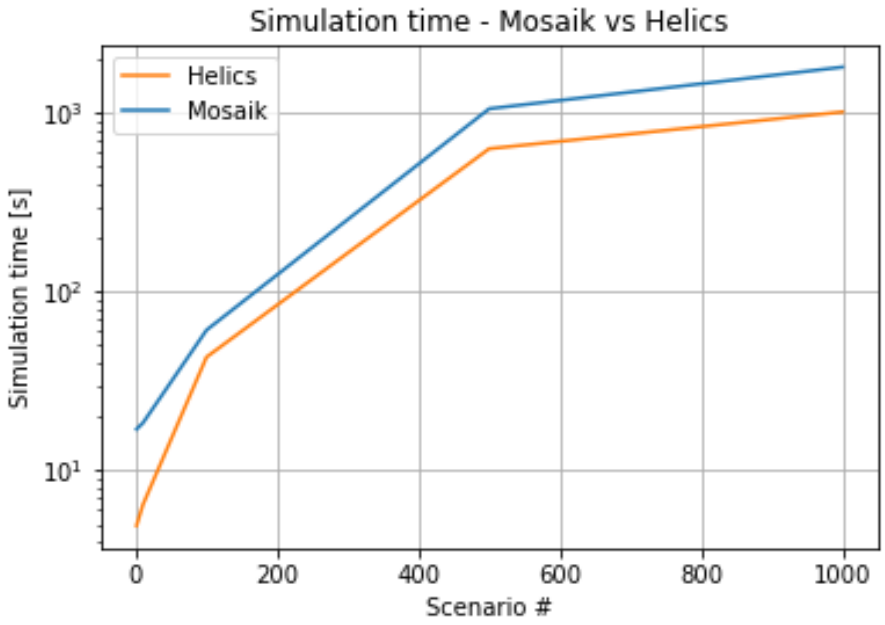

Figure 6.6: Simulation time Case 2

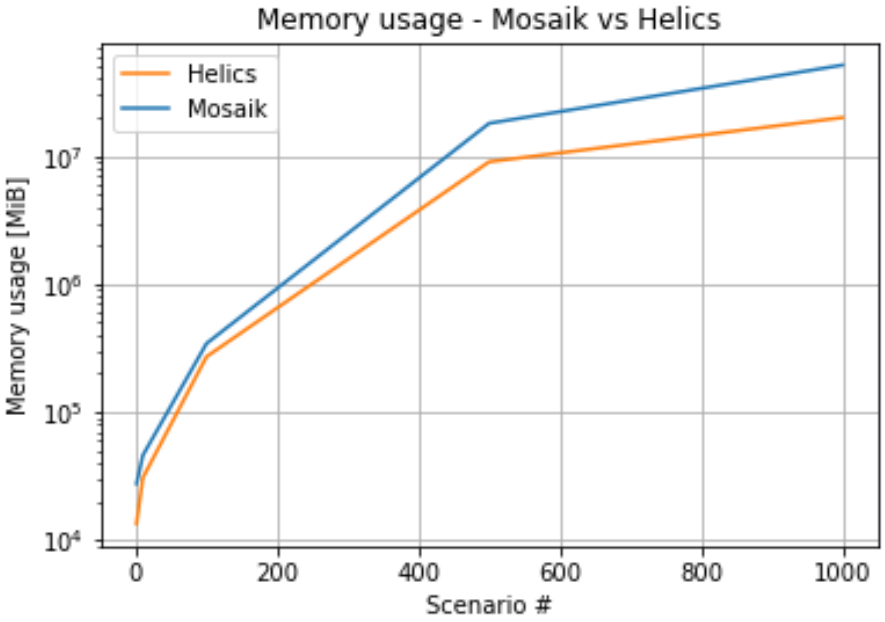

Figure 6.7: Memory usage Case 2

In a detailed view, figure 6.7 shows how in scenarios 1 and 10, the resources being used by both frameworks are very similar, but from scenario 100 onwards, the difference increases, making MOSAIK heavier. It can be inferred that as the scenarios continue to increase, so will the difference between the two curves.

Chapter 7

Conclusions

This thesis work provides a review of the MOSAIK and HELICS frameworks, with an emphasis on the differences and similarities that each has conceptually in the handling of information flow and the mechanism employed to synchronize and time step. For future research with simulation environments, this research serves as a reference for deciding which tool can best suit the requirements.

One of the similarities between MOSAIK and HELICS is found in the conceptual structure of the frameworks since both present a layered structure. Another similarity to point out is the methodology to be followed to implement a study scenario: In both MOSAIK and HELICS, the first step is to start by creating the API for each simulator/federate, then the configuration of the study case is defined together with the different interconnections, and finally, the co-simulation is executed. This process is the same regardless of the framework used.

Although the methodology is the same, its implementation differs from one framework to another. One of the differences is the interface (APIs) that must be implemented in each simulator/federate in order to connect the models. The MOSAIK API has three mandatory interface calls that must be developed for each one (init, create, step). Depending on whether the simulator supplies information to other simulators or has different processes to execute, additional calls must be developed. On the other hand, HELICS has a large set of interface calls that can be used in a versatile way providing more freedom in its development but at the same time translating into more work to implement them. Another difference is in the configuration of the scenario and connections. MOSAIK presents a single file called scenario script in which the input values of each element are configured, as well as the connections and data exchange that the scenario presents. on the contrary, HELICS establishes these configurations in the APIs of the federates.

The tests performed with the two case studies clearly showed that HELICS is faster and requires fewer computational resources than MOSAIK. It is worth mentioning that during the study, it became evident that the HELICS framework is still under development by its creators. At the beginning of the research, versions prior to 3.2 were used, which presented errors that caused the simulations to fail; with version 3.2, these problems were solved.

It was observed in the simulations that when there are scenarios with large numbers of simulators/federates, the differences in both time and resources required by MOSAIK increase significantly compared to the performance of HELICS. It is inferred that these advantages shown by HELICS are due to the structure it uses to synchronize federates and handle data flow with different communication protocol options. In summary, in large and extensive cases, it is more advantageous to use HELICS, and in small cases, either is a reasonable option.

Finally, the co-simulation system that allows the use of the MOSAIK and HELICS frameworks was tested in the two case studies. Using a YAML file as shown in the annexes, each case was configured along with its respective models. Subsequently, with the developed codes that support the simulations in both MOSAIK and HELICS, the systems were executed, allowing to validate that the simulations were executed correctly.

Appendix A

Appendix

A.1 Case 1: YAML file setup

code/Scenario_scheme_case1.yaml

```yaml
ORCHESTRATOR: "MOSAIK" # Choose between MOSAIK/HELICS
SCENARIO SCHEMA YAML:
 SCENARIO CONFIGURATION:
  - SCENARIO_NAME: "Case_one"
  - START_DATE: '1388534400'
  - DAYS: 7
  - BROKER_HOST: "192.168.236.69"
  - BROKER_PORT: "73100"
  - BROKER_NAME: broker
  - BROKER_KEY: "MosaikHelicsTesisJGRV"
  - NUMBER: 1
 SIMULATORS CONFIGURATION:
  Simulator:
   - SIMULATOR_NAME: "HouseholdSim"
     API_MOSAIK: "mosaik_householdsim"
     API_HELICS: "hhsim_api"
     RUN PROCESS:
      - MODE_RUN: "connect"
      - MODE_RUN_HELICS: "python"
      - HOSTPORT: "192.168.236.186"
      - PORT : "13200"
     PARAMETERS:
      - metadata : Full # Must be Empty in case the simulator need to write the metadata
      - stepTime: 60
      - timeAdvance: 0
     MODELS:
      - MODEL_NAME: ResidentialLoads
        NUMBER: 10
```

```
29          Model_instance:
30           - INSTANCE_NAME: "ResidentialLoads"
31           - PARAMS_NAME: ""
32           - OPT_METHOD: "children"
33    - SIMULATOR_NAME: "PyPower"
34      API_MOSAIK: "mosaik_pypower"
35      API_HELICS: "pypower_api"
36      BROKER_KEY: "MosaikHelicsTesisJGRV"
37      RUN PROCESS:
38       - MODE_RUN: "connect"
39       - MODE_RUN_HELICS: "python"
40       - HOSTPORT: "192.168.236.186"
41       - PORT : "32300"
42      PARAMETERS:
43       - metadata : Full # Must be Empty in case the simulator need to
        write the metadata
44       - step_size: 60
45       - timeAdvance: 60
46      MODELS:
47       - MODEL_NAME: Grid
48         NUMBER: 1
49          Model_instance:
50           - INSTANCE_NAME: "Grid"
51           - PARAMS_NAME: ""
52           - OPT_METHOD: "children"
53    - SIMULATOR_NAME: "CSV"
54      API_MOSAIK: "mosaik_csv"
55      API_HELICS: "PV_api"
56      BROKER_KEY: "MosaikHelicsTesisJGRV"
57      RUN PROCESS:
58       - MODE_RUN: "python"
59       - MODE_RUN_HELICS: "python"
60       - HOSTPORT: "192.168.236.69"
61       - PORT : "38500"
62      PARAMETERS:
63       - metadata : Full # Must be Empty in case the simulator need to
        write the metadata
64       - sim_start: '1388534400'
65       - datafile: 'pv_10kw.csv'
66       - stepTime: 60
67      MODELS:
68       - MODEL_NAME: "PV"
69         NUMBER: 10
70          Model_instance:
71           - INSTANCE_NAME: "PV_create"
72           - PARAMS_NAME: "PV_create"
73           - OPT_METHOD: "create"
74    - SIMULATOR_NAME: "MosaikHdf5" # Esta diferente al YAML de Helics,
      xq? En YAML Helics es DB en YAML Mosaik es MosaikHdf5
```

```
75        API_MOSAIK: "mosaik-hdf5"
76        API_HELICS: "hdf5_api"
77        BROKER_KEY: "MosaikHelicsTesisJGRV"
78        RUN PROCESS:
79         - MODE_RUN: "cmd"
80         - MODE_RUN_HELICS: "python"
81         - HOSTPORT: "192.168.236.69"
82         - PORT : "43600"
83        PARAMETERS:
84         - metadata : Full # Must be Empty in case the simulator need to
      write the metadata
85         - step_size: 60
86         - duration: 2
87         - timeAdvance: 60
88        MODELS:
89         - MODEL_NAME: "DB"
90           NUMBER: 1
91            Model_instance:
92             - INSTANCE_NAME: "Database"
93             - PARAMS_NAME: ""
94  CONNECTIONS:
95   - CONECTOR_NUMBER: 1
96     TYPE: "Many to One"
97     TOTAL_AMOUNT: 1
98     TOTAL_PUB: 1
99     TOTAL_SUB: 1
100    FROM: ResidentialLoads
101    TO: DB
102    ATTRIBUTES:
103     - ATTR: 'P_out'
104  - CONECTOR_NUMBER: 2
105    TYPE: "Nodes Many to One"
106    TOTAL_AMOUNT: 1
107    TOTAL_PUB: 1
108    TOTAL_SUB: 1
109    FROM: Grid
110    TO: DB
111    ATTRIBUTES:
112     - ATTR: 'P'
113     - ATTR: 'Q'
114     - ATTR: 'Vl'
115     - ATTR: 'Vm'
116     - ATTR: 'Va'
117  - CONECTOR_NUMBER: 3
118    TYPE: "Many to One"
119    TOTAL_AMOUNT: 1
120    TOTAL_PUB: 1
121    TOTAL_SUB: 1
122    FROM: PV
```

```
    TO: DB
    ATTRIBUTES:
     - ATTR: 'P'
  - CONECTOR_NUMBER: 4
    TYPE: "Branches Many to One"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: Grid
    TO: DB
    ATTRIBUTES:
     - ATTR: 'P_from'
     - ATTR: 'Q_from'
     - ATTR: 'P_to'
     - ATTR: 'P_from'
  - CONECTOR_NUMBER: 5
    TYPE: "Nodes Randomly"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: PV
    TO: Grid
    ATTRIBUTES:
     - ATTR: 'P'
  - CONECTOR_NUMBER: 6
    TYPE: "Building to grid"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: ResidentialLoads
    TO: Grid
    ATTRIBUTES:
     - ATTR: 'P_out'
     - ATTR: 'P'
```

A.2 Case 1: Models YAML file setup

code/Models_scheme_case1.yaml

```
MODELS:
  - MODEL_NAME: ResidentialLoads
    INPUT:
     - sim_start: '1388534400'
     - profile_file: 'profiles.data.gz'
     - grid_name: 'demo_lv_grid'
    Model instance:
     - INSTANCE_NAME: "ResidentialLoads"
```

```
      - PARAMS_SET: ""
    PARAMETERS:
      - PARAMS_SET_NAME: ""
        NUMBER: 1
  - MODEL_NAME: Grid
    INPUT:
      - gridfile: 'demo_lv_grid.json'
    Model instance:
      - INSTANCE_NAME: "Grid"
      - PARAMS_SET: ""
    PARAMETERS:
      - PARAMS_SET_NAME: ""
        NUMBER: 2
  - MODEL_NAME: PV
    INPUT:
    Model instance:
      - INSTANCE_NAME: "PV_create"
      - PARAMS_SET: "PV_create"
    PARAMETERS:
      - PARAMS_SET_NAME: "PV_create"
        NUMBER: 20
  - MODEL_NAME: DB
    INPUT:
      - filename: 'demo.hdf5'
      - info: 'yes'
    Model instance:
      - INSTANCE_NAME: "Database"
      - PARAMS_SET: "Empty"
    PARAMETERS:
      - PARAMS_SET_NAME: "Empty"
        NUMBER: 4
```

A.3 Case 2: YAML file setup

code/Scenario_scheme_case2.yaml

```
ORCHESTRATOR: "HELICS" # Choose the framework
SCENARIO SCHEMA YAML:
 SCENARIO CONFIGURATION:
  - SCENARIO_NAME: "Case-two"
  - START_DATE: '1420070400' #'2015-01-01 00:00:00'
  - DAYS: 7
  - BROKER_HOST: "192.168.236.69"
  - BROKER_PORT: "14800"
  - BROKER_NAME: broker
  - BROKER_KEY: "MosaikHelicsTesisJGRV"
  - NUMBER: 1
```

```
SIMULATORS CONFIGURATION:
  Simulator:
   - SIMULATOR_NAME: "AgentScheduler"
     API_MOSAIK: "mk_scheduler"
     API_HELICS: "hl_scheduler_api"
     RUN PROCESS:
      - MODE_RUN: "python"
      - MODE_RUN_HELICS: "python"
      - HOSTPORT: "192.168.236.69"
      - PORT : "20200"
     PARAMETERS:
      - metadata : Empty # Must be Empty in case the simulator need to write the metadata
      - days: 7
      - start_date: '2015-01-01 00:00:00'
     MODELS:
      - MODEL_NAME: "Scheduler"
        NUMBER: 1
        Model_instance:
         - INSTANCE_NAME: "Schedule"
         - PARAMS_NAME: "schedule"
   - SIMULATOR_NAME: "FMUAdapter"
     API_MOSAIK: "mk_fmu_pyfmi"
     API_HELICS: "hl_fmu_pyfmi_api"
     RUN PROCESS:
      - MODE_RUN: "python"
      - MODE_RUN_HELICS: "python"
      - HOSTPORT: "192.168.236.69"
      - PORT : "22200"
     PARAMETERS:
      - metadata : Empty # Must be Empty in case the simulator need to write the metadata
      - step_size: 600
     MODELS:
      - MODEL_NAME: "HeatPump"
        NUMBER: 1
        Model_instance:
         - INSTANCE_NAME: "HeatPump"
         - PARAMS_NAME: "heatpump"
   - SIMULATOR_NAME: "FMUAdapter"
     API_MOSAIK: "mk_fmu_pyfmi"
     API_HELICS: "hl_fmu_pyfmi_api"
     RUN PROCESS:
      - MODE_RUN: "python"
      - MODE_RUN_HELICS: "python"
      - HOSTPORT: "192.168.236.69"
      - PORT : "23200"
     PARAMETERS:
```

```
58        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
59        - step_size: 600 #
60        - stop_time: 31536000 #
61      MODELS:
62        - MODEL_NAME: "Building"
63          NUMBER: 1
64          Model_instance:
65           - INSTANCE_NAME: "Building"
66           - PARAMS_NAME: "building"
67    - SIMULATOR_NAME: "PVSim"
68      API_MOSAIK: "mk_pvsim"
69      API_HELICS: "hl_pv_api"
70      RUN PROCESS:
71        - MODE_RUN: "connect"
72        - MODE_RUN_HELICS: "python"
73        - HOSTPORT: "192.168.236.186"
74        - PORT : "24200"
75      PARAMETERS:
76        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
77        - step_size: 600 #
78        - start_date: '2015-01-01 00:00:00'
79      MODELS:
80        - MODEL_NAME: PV
81          NUMBER: 1
82          Model_instance:
83           - INSTANCE_NAME: "PV"
84           - PARAMS_NAME: "pv"
85    - SIMULATOR_NAME: "Meter"
86      API_MOSAIK: "mk_metersim"
87      API_HELICS: "hl_meter_api"
88      RUN PROCESS:
89        - MODE_RUN: "connect"
90        - MODE_RUN_HELICS: "python"
91        - HOSTPORT: "192.168.236.186"
92        - PORT : "25200"
93      PARAMETERS:
94        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
95        - step_size: 600
96        - collect_data: false
97      MODELS:
98        - MODEL_NAME: Nodo
99          NUMBER: 1
100         Model_instance:
101          - INSTANCE_NAME: "Nodo"
102          - PARAMS_NAME: "nodo"
103   - SIMULATOR_NAME: "Meter"
```

```
104       API_MOSAIK: "mk_metersim"
105       API_HELICS: "hl_meter_api"
106       RUN PROCESS:
107        - MODE_RUN: "connect"
108        - MODE_RUN_HELICS: "python"
109        - HOSTPORT: "192.168.236.186"
110        - PORT : "26200"
111       PARAMETERS:
112        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
113        - step_size: 600
114        - collect_data: false
115       MODELS:
116        - MODEL_NAME: SM
117          NUMBER: 1
118           Model_instance:
119            - INSTANCE_NAME: "SM"
120            - PARAMS_NAME: "sm"
121     - SIMULATOR_NAME: "CSV"
122       API_MOSAIK: "mk_csvsim"
123       API_HELICS: "hl_csv_api"
124       RUN PROCESS:
125        - MODE_RUN: "connect"
126        - MODE_RUN_HELICS: "python"
127        - HOSTPORT: "192.168.236.186"
128        - PORT : "27200"
129       PARAMETERS:
130        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
131        - start_date: '2015-01-01 00:00:00'
132        - datafile: 'timeseries/famiglia.csv'
133       MODELS:
134        - MODEL_NAME: Home
135          NUMBER: 1
136           Model_instance:
137            - INSTANCE_NAME: "Home"
138            - PARAMS_NAME: "Home_create"
139            - OPT_METHOD: "create"
140     - SIMULATOR_NAME: "CSV"
141       API_MOSAIK: "mk_csvsim"
142       API_HELICS: "hl_csv_api"
143       RUN PROCESS:
144        - MODE_RUN: "python"
145        - MODE_RUN_HELICS: "python"
146        - HOSTPORT: "192.168.236.69"
147        - PORT : "28200"
148       PARAMETERS:
149        - metadata : Empty # Must be Empty in case the simulator need
      to write the metadata
```

```
      - start_date: '2015-01-01 00:00:00'
      - datafile: 'timeseries/meteo_mosaik.csv'
    MODELS:
      - MODEL_NAME: Meteo
        NUMBER: 1
        Model_instance:
          - INSTANCE_NAME: "Meteo"
          - PARAMS_NAME: "Meteo_create"
          - OPT_METHOD: "create"
CONNECTIONS:
  - CONECTOR_NUMBER: 1
    TYPE: "Direct"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: Building
    TO: HeatPump
    ATTRIBUTES:
      - ATTR: ['TRooMea', 'TroomSens']
      - ATTR: 'Tamb'
  - CONECTOR_NUMBER: 2
    TYPE: "Direct"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: HeatPump
    TO: Nodo
    ATTRIBUTES:
      - ATTR: ['Power', 'Load']
  - CONECTOR_NUMBER: 3
    TYPE: "Many to One"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: Home
    TO: Nodo
    ATTRIBUTES:
      - ATTR: ['P', 'Load']
  - CONECTOR_NUMBER: 4
    TYPE: "Direct"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: Nodo
    TO: SM
    ATTRIBUTES:
      - ATTR: ['Pload', 'Load']
      - ATTR: ['Pprod', 'Prod']
  - CONECTOR_NUMBER: 5
```

```
    TYPE: "Direct"
    TOTAL_AMOUNT: 1
    TOTAL_PUB: 1
    TOTAL_SUB: 1
    FROM: Schedule
    TO: HeatPump
    ATTRIBUTES:
     - ATTR: 'Tset'
```

A.4 Case 2: Models YAML file setup

code/Models_scheme_case2.yaml

```
MODELS:
  - MODEL_NAME: "Scheduler"
    INPUT:
     - PUBLIC: True
     - ATTRS: ""
     - start_date: "1420070400"
    Model instance:
     - INSTANCE_NAME: "Schedule"
     - PARAMS_SET: "schedule"
     - ATTRS: [Tset]
    PARAMETERS:
     - PARAMS_SET_NAME: "schedule"
       schedule: ['0:16','6:20','22:16']#
    ['0:16','4:18','7:20',"19:18",'22:16']
  - MODEL_NAME: "Battery"
    INPUT:
     - PUBLIC: True
     - ATTRS: ""
     - fmu_class: "battery"
     - solver: "matlab"
     - params: "[LoadINW,PnetBatt]"
     - step_size: 600
    Model instance:
     - INSTANCE_NAME: "Battery"
     - PARAMS_SET: "battery"
     - ATTRS: [LoadINW, GenINW, I, V, SOC, PnetBatt]
    PARAMETERS:
     - PARAMS_SET_NAME: "battery"
       fmu_name: battery3
       instance_name: ['0']
       start_vrs: []
       start_in_vrs: []
  - MODEL_NAME: "HeatPump"
    INPUT:
```

```
    - PUBLIC: True
    - ATTRS: ""
    - fmu_class: "heatpump"
    - solver: "empty"
    - step_size: 600
   Model instance:
    - INSTANCE_NAME: "HeatPump"
    - PARAMS_SET: "heatpump"
    - ATTRS: [Ttank, Tr,Tm, Tset, Tamb, Power, COP, Qsensible, MV,
  Td, TroomSens]
   PARAMETERS:
    - PARAMS_SET_NAME: "heatpump"
      fmu_name: HeatPump08112021 #HeatPumpv12aw
      instance_name: [HP]
      start_vrs: {contr_type: "PID",
      AW: 1.5,
      Td: 3000,
      Ti: 1500,
      k: 0.003,
      Lp: 1200,
      DiamTubes: 1,
      Ufloor: 2,
      G_water_nominal: 2,
      yMax: 1,
      QHPmax: 12000,
      GHP: 0.05,
      V: 300}
      start_in_vrs: {Tset: 16,Ttank: 30}
- MODEL_NAME: "Building"
   INPUT:
    - PUBLIC: True
    - ATTRS: ""
    - fmu_class: "building"
    - solver: "empty"
    - step_size: 600
   Model instance:
    - INSTANCE_NAME: "Building"
    - PARAMS_SET: "building"
    - ATTRS: [Q, Peo, TRooMea, Tamb, Tdew, Tbulb, RH]
   PARAMETERS:
    - PARAMS_SET_NAME: "building"
      fmu_name: building_tia
      instance_name: [Home] # in brackets
      start_vrs: {}
      start_in_vrs: {}
- MODEL_NAME: "PV"
   INPUT:
    - PUBLIC: True
    - ATTRS: ""
```

```
82        - start_date: "1420070400"
83        - step_size: 600
84      Model instance:
85        - INSTANCE_NAME: "PV"
86        - PARAMS_SET: "pv"
87        - ATTRS: [power_dc, ghi, T_ext]
88      PARAMETERS:
89        - PARAMS_SET_NAME: "pv_sim"
90          P_system: 5000
91          slope: 35
92          aspect: 0
93          latitude: 45.7
94          longitude: 7.6
95          elevation: 230
96    - MODEL_NAME: "Nodo"
97      INPUT:
98        - PUBLIC: True
99        - ATTRS: ""
100       - step_size: 600
101     Model instance:
102       - INSTANCE_NAME: "Nodo"
103       - PARAMS_SET: "nodo"
104       - ATTRS: [Load, Prod, NetBatt, Pnet, Pprod, Pload, Pnetbatt,
        Pexport, Pinport]
105     PARAMETERS:
106       - PARAMS_SET_NAME: "nodo"
107         # empty: []
108   - MODEL_NAME: "SM"
109     INPUT:
110       - PUBLIC: True
111       - ATTRS: ""
112       - step_size: 600
113     Model instance:
114       - INSTANCE_NAME: "SM"
115       - PARAMS_SET: "sm"
116       - ATTRS: [Load, Prod, NetBatt, Pnet, Pprod, Pload, Pnetbatt,
        Pexport, Pinport]
117     PARAMETERS:
118       - PARAMS_SET_NAME: "sm"
119         # empty: []
120   - MODEL_NAME: "Home"
121     INPUT:
122       - PUBLIC: True
123       - ATTRS: ""
124       - step_size: 600
125       - sim_start: '1388534400'
126       - datafile: 'timeseries/famiglia.csv'
127     Model instance:
128       - INSTANCE_NAME: "Home"
```

```
      - PARAMS_SET: "Home_create"
      - ATTRS: [Pres, P]
    PARAMETERS:
      - PARAMS_SET_NAME: "Home_create"
        NUMBER: 1
  - MODEL_NAME: "Meteo"
    INPUT:
      - PUBLIC: True
      - ATTRS: ""
      - step_size: 600
      - sim_start: '1388534400'
      - datafile: 'timeseries/meteo_mosaik.csv'
    Model instance:
      - INSTANCE_NAME: "Meteo"
      - PARAMS_SET: "Meteo_create"
      - ATTRS: [ghi, T_ext, T_dewPoint, humidity]
    PARAMETERS:
      - PARAMS_SET_NAME: "Meteo_create"
        NUMBER: 1
```

A.5 Backend Mosaik simulation

code/opt_scenario_mosaik.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 27 10:00:18 2022

@author: Juan
"""
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 18 18:10:05 2021

@author: Juan
"""
import json
import yaml
import time
import copy
import os
import sys
import random
import mosaik
from mosaik.util import connect_randomly, connect_many_to_one
from datetime import datetime
from pathlib import Path
```

```
24 import socket
25 import subprocess
26
27 hostname=socket.gethostname()
28 ref=socket.gethostbyname(hostname)
29
30
31 #
    ==========================================================================

32 # Loading YAML files
33 #
    ==========================================================================

34 main_Path = os.path.normpath(os.getcwd() + os.sep + os.pardir).
    replace('\\', '/')
35 dirs = os.listdir(main_Path)
36 os.chdir(main_Path)
37
38 for file in dirs:
39     check = file.split(".")
40     if len(check) == 1:
41         pass
42     else:
43         if check[1] == 'yaml':
44             if file.split("_")[0] == 'scenario':
45                 scenario = str(file)
46             if file.split("_")[0] == 'Models':
47                 modelos = str(file)
48         else:pass
49
50 #
    ==========================================================================

51 # Opening YAML files
52 #
    ==========================================================================

53 with open(scenario) as file:
54     data = yaml.load(file, Loader=yaml.FullLoader)
55
56 with open(modelos) as file:
57     models = yaml.load(file, Loader=yaml.FullLoader)
58
59
60 # Back to main path
61 current_Path = os.getcwd().replace('\\', '/')
62 main_Path = current_Path+'/Mosaik'
63 os.chdir(main_Path)
```

```
64
65 #
    ============================================================================

66 # Starting scenario configration
67 #
    ============================================================================

68
69 n = len(data['SCENARIO SCHEMA YAML']['SIMULATORS CONFIGURATION']['
    Simulator'])
70 sim = data['SCENARIO SCHEMA YAML']['SIMULATORS CONFIGURATION']['
    Simulator']
71 sce = data['SCENARIO SCHEMA YAML']['SCENARIO CONFIGURATION']
72 num_sce = 1
73
74 for i in range(len(sce)):
75     for k,v in sce[i].items():
76         if k == 'DAYS':
77             END = int(v)*24*3600
78         else:
79             END = 1*24*3600
80     for k,v in sce[i].items():
81         if k == 'NUMBER':
82             num_sce = int(v)
83
84 # print('los escenarios son:', num_sce)
85
86 #
    ============================================================================

87 # Importando todas las APIs de Mosaik
88 #
    ============================================================================

89 sys.path.append(os.getcwd().replace('\\', '/')+'/Api_Mosaik')
90
91 for i in range(len(sim)):
92     var = str(sim[i]['API_MOSAIK'].split(':')[0])
93     if var == 'mosaik-hdf5':
94         pass
95     else:
96         exec('import '+var)
97     # print(var)
98
99 #
    ============================================================================

100 # Agregando a las sim las cantidades de simuladores solictados
```

```
101 #
    ================================================================================

102 mydata = copy.deepcopy(sim)
103 for s in range(len(mydata)):
104     num = mydata[s]['MODELS'][0]['NUMBER']
105     if num > 1:
106         for z in range(int(num)-1):
107             mydata.append(sim[s])
108
109 #
    ================================================================================

110 # Escribiendo el archivo sim_config que contiene la información de
    los simuladores a usar
111 #
    ================================================================================

112 ot_sim = []
113 sim_ip = []
114 sim_config = {}
115 for i in range(len(mydata)):
116     s_num = 0
117     sim_ip.append(str(mydata[i]['RUN PROCESS'][3]['PORT']))
118     for tr in ot_sim:
119             if tr.split("_")[0] == mydata[i]['SIMULATOR_NAME']:
120                 s_num += 1
121     new_nombre = str(mydata[i]['SIMULATOR_NAME'])+'_'+str(s_num)
122     ot_sim.append(new_nombre)
123     if mydata[i]['RUN PROCESS'][0]['MODE_RUN'] == 'python':
124         sim_config[new_nombre] = {mydata[i]['RUN PROCESS'][0]['
    MODE_RUN']: str(mydata[i]['API_MOSAIK'])+':'+str(mydata[i]['
    SIMULATOR_NAME'])}
125     elif mydata[i]['RUN PROCESS'][0]['MODE_RUN'] == 'cmd':
126         sim_config[new_nombre] = {mydata[i]['RUN PROCESS'][0]['
    MODE_RUN']: str(mydata[i]['API_MOSAIK'])+' %(addr)s'}
127     elif mydata[i]['RUN PROCESS'][0]['MODE_RUN'] == 'connect':
128         nw_port = int(mydata[i]['RUN PROCESS'][3]['PORT'])+int(s_num)
129         sim_config[new_nombre] = {mydata[i]['RUN PROCESS'][0]['
    MODE_RUN']:(str(mydata[i]['RUN PROCESS'][2]['HOSTPORT'])+':'+str(
    nw_port))}
130
131 print('line 97 - sim_config es:', sim_config)
132
133 #
    ================================================================================

134 # Taking into account IP and Port
```

```
135 #
    ================================================================================

136 for q in sce:
137     if 'BROKER_HOST' in (q.keys()):
138         central_addres = q['BROKER_HOST']
139
140 total_info = []
141 for g in range(len(sim)):
142     for nu in range(sim[g]['MODELS'][0]['NUMBER']):
143         add_info = {'api':'',
144             'ip':'',
145             'port':''}
146         add_info['api'] = sim[g]['API_MOSAIK']
147         for j in sim[g]['RUN PROCESS']:
148             if 'HOSTPORT' in list(j.keys()):
149                 add_info['ip'] = j['HOSTPORT']
150             if 'PORT' in list(j.keys()):
151                 num_port = int(j['PORT'])+int(nu)
152                 add_info['port'] = str(num_port)
153         total_info.append(add_info)
154
155 def main():
156     random.seed(23)
157     world = mosaik.World(sim_config)
158     for n_sce in range(num_sce):
159         create_scenario(world)
160     world.run(until=END)  # As fast as possilbe
161
162 def create_scenario(world):
163     """ACÁ ESTOY INICIANDO EL CREATE SCENARIO"""
164
165     for s in range(len(sim)):
166         num = sim[s]['MODELS'][0]['NUMBER']
167         if num > 1:
168             for z in range(int(num)-1):
169                 sim.append(sim[s])
170
171     sim_name = []
172     for i in range(len(sim)):
173         sim_param = {}
174         key_param = []
175         value_param = []
176         for m in range(len(sim[i]['MODELS'])):
177             if not sim[i]['PARAMETERS']: # Condición no vienen
    parametros definidos
178                 globals()[sim[i]['SIMULATOR_NAME']] = world.start(sim
    [i]['SIMULATOR_NAME'])
179                 print('No hay parameters')
```

```
180             else: # Condición cuando sí vienen parametros definidos
181                 for p in range(len(sim[i]['PARAMETERS'])):
182                     for k,v in  sim[i]['PARAMETERS'][p].items():
183                         if k == 'metadata':
184                             if v == 'Empty': # Condición para escribir la variable metadata
185                                 metadata = eval(sim[i]['API_MOSAIK'].split(':')[0]+'.META')
186                                 for t in range (len(models['MODELS'])):
187                                     if sim[i]['MODELS'][m]['MODEL_NAME'] == models['MODELS'][t]['MODEL_NAME']:
188                                         if sim[i]['MODELS'][m]['Model_instance'][0]['INSTANCE_NAME'] == models['MODELS'][t]['Model instance'][0]['INSTANCE_NAME']:
189                                             metadata['models'] = {models['MODELS'][t]['Model instance'][0]['INSTANCE_NAME']:{'public': True,
190 
                                                                                                                                'params': '',
191 
                                                                                                                                'attrs': models['MODELS'][t]['Model instance'][2]['ATTRS']}}
192                                         if sim[i]['MODELS'][m]['Model_instance'][1]['PARAMS_NAME'] == models['MODELS'][t]['Model instance'][1]['PARAMS_SET']:
193                                             params = []
194                                             for k,v in models['MODELS'][t]['PARAMETERS'][0].items():
195                                                 if k == 'PARAMS_SET_NAME':
196                                                     pass
197                                                 elif k == 'step_size':
198                                                     pass
199                                                 elif k == 'timeAdvance':
200                                                     pass
201                                                 else:
202                                                     params.append(k) # Guardar los parametros a incluir en el metadata
203                                             metadata['models'][models['MODELS'][t]['Model instance'][0]['INSTANCE_NAME']]['params'] = params
204                                 # print(metadata) # Verificando si la variable metadata se escribió bien o no
205                                 sim_param['sim_meta'] = metadata
206                     for key,value in sim[i]['PARAMETERS'][p].items():
```

```
207                         if key == 'timeAdvance':
208                             pass
209                         elif key == 'stepTime':
210                             pass
211                         elif key == 'metadata':
212                             pass
213                         elif key == 'sim_start':
214                             key_param.append(key)
215                             value_param.append(str(datetime.
    fromtimestamp(int(value))))
216                             sim_param[key] = str(datetime.
    fromtimestamp(int(value)))
217                         elif key == 'duration':
218                             key_param.append(key)
219                             value_param.append(int(value)*24*3600)
220                             sim_param[key] = int(value)*24*3600
221                         elif key == 'days':
222                             key_param.append(key)
223                             value_param.append(int(value))
224                             sim_param[key] = int(value)
225                         elif key == 'start_date':
226                             key_param.append(key)
227                             value_param.append(str(value))
228                             sim_param[key] = str(value)
229                         elif key == 'step_size':
230                             key_param.append(key)
231                             value_param.append(int(value))
232                             sim_param[key] = int(value)
233                         else:
234                             key_param.append(key)
235                             value_param.append(value)
236                             sim_param[key] = value
237                 name = sim[i]['SIMULATOR_NAME'].replace("_","")
238                 n_name = 0
239                 for s in sim_name:
240                     if s.split("_")[0] == name:
241                         n_name += 1
242                 sim_nombre = str(name)+'_'+str(n_name)
243                 sim_name.append(sim_nombre)
244                 print('name',sim_nombre)
245                 body = 'world.start("%s",**sim_param)'%str(sim_nombre
    )
246                 print('line',body)
247                 print('line 189: los sim param son:', sim_param)
248                 print('linea Betto INTERES:',sim[i]['MODELS'][0]['
    NUMBER'])
249                 globals()[sim_nombre] = eval(body)
250                 for k,v in sim_param.items():
251                     if k == 'solver':
```

```
252                         body = sim_nombre+'.solver_call('+str(k)+'='+
    "'%s', num=%i)"%(str(v),int(sim[i]['MODELS'][0]['NUMBER']))
253                         print('solver call',body)
254                         eval(body)
255                         # battery.solver_call(solver='matlab')
256                     else:
257                         pass
258
259
260     #
    ================================================================================

261     # Versión nueva del código
262     #
    ================================================================================

263
264     # Creando los models
265     sim_name_dos = []
266     mod_name = []
267     for i in range(len(sim)):
268         method_opt = False
269         var_method_opt = 'nothing'
270         model = ''
271         name_dos = sim[i]['SIMULATOR_NAME'].replace("_","")
272         n_name_dos = 0
273         for s in sim_name_dos:
274             if s.split("_")[0] == name_dos:
275                 n_name_dos += 1
276         sim_nombre_dos = str(name_dos)+'_'+str(n_name_dos)
277         sim_name_dos.append(sim_nombre_dos)
278         for m in range(len(sim[i]['MODELS'])):
279             model = sim[i]['MODELS'][m]['MODEL_NAME']
280             # if model != 'ResidentialLoads' and model != 'Grid':
281             for t in range(len(models['MODELS'])):
282                 # Revisar si el nombre del Modelo es el mismo
283                 if models['MODELS'][t]['MODEL_NAME'] == model:
284                     # print('el model aquí esta ---------:', model)
285                     n_mod_name = 0
286                     for r in mod_name:
287                         if r.split("_")[0] == model:
288                             n_mod_name += 1
289                     mod_nombre = str(model)+'_'+str(n_mod_name)
290                     mod_name.append(mod_nombre)
291                     print('el model aquí esta ----------:', mod_nombre)
292                     # print(models['MODELS'][t]['MODEL_NAME'], model)
293                     new_name = sim[i]['MODELS'][m]['Model_instance'
    ][0]['INSTANCE_NAME']
294                     num = 1
```

```
295                     follow = False
296                     instan = False
297                 # Revisar si el nombre de la Instancia es la misma
298                     for n in range(len(models['MODELS'][t]['Model
    instance'])):
299                         for k,v in models['MODELS'][t]['Model
    instance'][n].items():
300                             if k == 'INSTANCE_NAME':
301                                 for z in range(len(sim[i]['MODELS'][m
    ]['Model_instance'])):
302                                     for key,val in sim[i]['MODELS'][m
    ]['Model_instance'][z].items():
303                                         if key == 'INSTANCE_NAME':
304                                             if val == v:
305                                                 # print(val)
306                                                 follow = True
307                                                 # Revisando cual es
    el valor de la variable OPT_METHOD
308                                                 for p in sim[i]['
    MODELS'][m]['Model_instance']:
309                                                     for w in p:
310                                                         if w == '
    OPT_METHOD':
311                                                             if p[w]
    != 'nothing':
312
    method_opt = True
313
    var_method_opt = p[w]
314                                                             # print(p
    [w])
315                         # Revisar si el nombre de los Parámetros es
    el mismo
316                         for k,v in models['MODELS'][t]['Model
    instance'][n].items():
317                             if follow == True and k == 'PARAMS_SET':
318                                 for z in range(len(sim[i]['MODELS'][m
    ]['Model_instance'])):
319                                     for key,val in sim[i]['MODELS'][m
    ]['Model_instance'][z].items():
320                                         if key == 'PARAMS_NAME':
321                                             if val == v:
322                                                 name = v
323                                                 instan = True
324                     if method_opt == False or var_method_opt == '
    create':
325                         if instan == False and follow == True:
326                             if not models['MODELS'][t]['INPUT']:
327                                 print('Aquí estoy')
```

```
328                                 else:
329                                     inp = models['MODELS'][t]['INPUT']
330                                     for r in range(len(inp)):
331                                         if list(inp[r].keys())[0] == '
    info':
332                                             pass
333                                         else:
334                                             inp_param = inp[r]
335                                         # print(inp_param)
336                                         # inp_param = inp[i]
337                                     for n in range(len(models['MODELS'][t
    ]['Model instance'])):
338                                         for k,v in models['MODELS'][t]['
    Model instance'][n].items():
339                                             if k == 'INSTANCE_NAME':
340                                                 print('i es:',i)
341                                                 print('m es:',m)
342                                                 for z in range(len(sim[i
    ]['MODELS'][m]['Model_instance'])):
343                                                     print('numero',z)
344                                                     print('variable',sim[
    i]['MODELS'][m]['Model_instance'][z])
345                                                     for key,val in sim[i
    ]['MODELS'][m]['Model_instance'][z].items():
346                                                         if key == '
    INSTANCE_NAME':
347                                                             name_inst = v
348                                     beto = str(sim_nombre_dos)+'.'+str(
    name_inst)+'('+'**inp_param)'
349                                     print('los inp_param son:', inp_param
    )
350                                     print('linea 260- body no Grid-Resid:
    ',beto)
351                                     print('linea 261 - Name model no Grid
    -Resid:',mod_nombre)
352                                     globals()[mod_nombre] = eval(beto)
353
354                             if instan == True:
355                                 new_param = {}
356                                 for z in range(len(models['MODELS'][t]['
    PARAMETERS'])):
357                                     if models['MODELS'][t]['
    PARAMETERS'][z]['PARAMS_SET_NAME'] == name:
358                                         for k,v in models['MODELS'][t
    ]['PARAMETERS'][z].items():
359                                             if k == 'PARAMS_SET_NAME'
    :
360                                                 pass
361                                             elif k == 'NUMBER':
```

```
362                                                 new_param = int(v)
363                                             else:
364                                                 new_param[k] = v
365
366                                 if type(new_param) is dict:
367                                     body = str(sim_nombre_dos) + '.' +
    str(new_name) + '(**new_param)'
368                                     print('line 289 - body:', body)
369                                     print('linea 290 - model es:',
    mod_nombre)
370                                     globals()[mod_nombre] = eval(body)
371                                     # globals()[sim[i]['MODELS'][m]['
    Model_instance'][0]['INSTANCE_NAME']] = eval(body)
372                                 elif type(new_param) is int:
373                                     # var_global = str(sim[i]['MODELS'][m
    ]['MODEL_NAME'])
374                                     var_global = mod_nombre
375                                     print('line 294 - var global:',
    var_global)
376                                     # body = str(sim[i]['MODELS'][m]['
    MODEL_NAME']) + '.' + str(sim[i]['MODELS'][m]['MODEL_NAME']) +
    '.'+ str(new_name.split('_')[0]).lower() + '(new_param)'
377                                     body = str(sim_nombre_dos) + '.' +
    str(sim[i]['MODELS'][m]['MODEL_NAME']) + '.'+ str(var_method_opt).
    lower() + '(new_param)'
378                                     print('line 297 - Model no Grid-Resid
    :',mod_nombre)
379                                     print('line 298 - body is:',body)
380                                     print('linea 299:', new_param)
381                                     globals()[mod_nombre] = eval(body)
382                                     # globals()[sim[i]['MODELS'][m]['
    MODEL_NAME']] = eval(body)
383                                     # print('estamos aquí')
384                     else:
385                         for n in range(len(models['MODELS'])):
386                             if models['MODELS'][n]['MODEL_NAME'] ==
    model:
387                                 if not models['MODELS'][n]['INPUT']:
388                                     mod_inst = ""
389                                     for w in range(len(models['MODELS
    '][n]['Model instance'])):
390                                         for k,v in models['MODELS'][n
    ]['Model instance'][w].items():
391                                             if k == 'INSTANCE_NAME':
392                                                 if v == "":
393                                                     mod_inst = ""
394                                                 else:
395                                                     mod_inst= str(v).
    lower()
```

```
396                                                 if k == 'PARAMS_SET':
397                                                     if v != '':
398                                                         mode_met_param =
    str(v)
399                                                         # print(
    mode_met_param)
400                                                     else:
401                                                         mode_met_param =
    ""
402                                                         # print('Vacio
    betto')
403                                             if mode_met_param != "":
404                                                 param = []
405                                                 for p in range(len(models['
    MODELS'][n]['PARAMETERS'])):
406                                                     if models['MODELS'][n]['
    PARAMETERS'][p]['PARAMS_SET_NAME'] == mode_met_param:
407                                                         for key, val in
    models['MODELS'][n]['PARAMETERS'][p].items():
408                                                             if key == '
    PARAMS_SET_NAME':
409                                                                 pass
410                                                             else:
411                                                                 param.append(
    val)
412                                                 # beto = str(sim[i]['
    SIMULATOR_NAME'])+'.'+str(model[t])+'.'+str(mod_inst)+'(%d)'%param
    [0]
413                                                 beto = str(sim_nombre_dos)+'.
    '+str(model)+'.'+str(var_method_opt)+'(%d)'%param[0]
414                                                 print('body model:', beto) #
    Acá debo definir la linea de python a ejecutar de la instancia
415                                                 print('Model no Grid-Resid:',
    mod_nombre)
416                                                 globals()[mod_nombre] = eval(
    beto)
417                                                 # globals()[models['MODELS'][
    n]['MODEL_NAME']] = eval(beto)
418                                             else:
419                                                 # beto = str(sim[i]['
    SIMULATOR_NAME'])+'.'+str(model[t])+'.'+str(mod_inst)+'()'
420                                                 beto = str(sim_nombre_dos)+'.
    '+str(model)+'.'+str(var_method_opt)+'()'
421                                                 print('body model:', beto)
422                                                 print('Model no Grid-Resid:',
    mod_nombre)
423                                                 globals()[mod_nombre] = eval(
    beto)
```

```
424                                         # globals()[models['MODELS'][
    n]['MODEL_NAME']] = eval(beto)
425                                 else:
426                                     mod_param = {}
427                                     for p in range(len(models['MODELS
    ']][n]['INPUT'])):
428                                         for key,value in models['
    MODELS'][n]['INPUT'][p].items():
429                                             if key == 'sim_start':
430                                                 mod_param[key] = str(
    datetime.fromtimestamp(int(value)))
431                                             elif key == 'info':
432                                                 pass
433                                             else:
434                                                 mod_param[key] =
    value
435                                     mod_inst = ""
436                                     for w in range(len(models['MODELS
    ']][n]['Model instance'])):
437                                         for k,v in models['MODELS'][n
    ]['Model instance'][w].items():
438                                             if k == 'INSTANCE_NAME':
439                                                 if v == "":
440                                                     mod_inst = ""
441                                                 else:
442                                                     mod_inst= str(v).
    lower()
443                                             if k == 'PARAMS_SET':
444                                                 if v != '':
445                                                     mode_met_param =
    str(v)
446                                                     # print(
    mode_met_param)
447                                                 else:
448                                                     mode_met_param =
    ""
449                                                     # print('Vacio
    betto')
450                                     if mod_inst != "":
451                                         if mode_met_param != "":
452                                             param = []
453                                             for p in range(len(models
    ['MODELS'][n]['PARAMETERS'])):
454                                                 if models['MODELS'][n
    ]['PARAMETERS'][p]['PARAMS_SET_NAME'] == mode_met_param:
455                                                     for key, val in
    models['MODELS'][n]['PARAMETERS'][p].items():
456                                                         if key == '
    PARAMS_SET_NAME':
```

```
457                                                                 pass
458                                                             else:
459                                                                 param.
    append(val)
460                                                 # print(param)
461                                                 # print(str(param[0]))
462                                                 # beto = str(sim[i]['
    SIMULATOR_NAME'])+'.'+str(model[t])+'(**mod_param).'+str(mod_inst)
    +'(%s)'%str(param[0])
463                                                 beto = str(sim_nombre_dos
    )+'.'+str(model)+'(**mod_param).'+str(var_method_opt)+'(%s)'%str(
    param[0])
464                                                 print('body model:', beto
    )#Acá debo definir la linea de python a ejecutar de la instancia
465                                                 print('linea 383 - ins es
    :',mod_nombre)
466                                                 globals()[mod_nombre] =
    eval(beto)
467                                                 # globals()[models['
    MODELS'][n]['MODEL_NAME']] = eval(beto)
468                                             else:
469                                                 # beto = str(sim[i]['
    SIMULATOR_NAME'])+'.'+str(model[t])+'(**mod_param).'+str(mod_inst)
470                                                 beto = str(sim_nombre_dos
    )+'.'+str(model)+'(**mod_param).'+str(var_method_opt)
471                                                 print('body model:', beto
    )
472                                                 print('linea 388 - ins es
    :',mod_nombre)
473                                                 globals()[mod_nombre] =
    eval(beto)
474                                                 # globals()[models['
    MODELS'][n]['MODEL_NAME']] = eval(beto)
475                                         else:
476                                             # beto = str(sim[i]['
    SIMULATOR_NAME'])+'.'+str(model[t])+'(**mod_param)'
477                                             beto = str(sim_nombre_dos)+'.
    '+str(model)+'(**mod_param)'
478                                             print('body model:', beto)
479                                             print('linea 395 - ins es:',
    mod_nombre)
480                                             globals()[mod_nombre] = eval(
    beto)
481                                             # globals()[models['MODELS'][
    n]['MODEL_NAME']] = eval(beto)
482
483
484         print(method_opt, var_method_opt)
485
```

```
486
487     # Connections
488     if data['SCENARIO SCHEMA YAML']['CONNECTIONS'] is not None:
489         base_data = mod_name
490         con = data['SCENARIO SCHEMA YAML']['CONNECTIONS']
491         for i in range(len(con)):
492             if con[i]['TYPE'] == 'Many to One':
493                 hacia = con[i]['TO']
494                 desde = con[i]['FROM']
495                 # Cheking the FROM model
496                 if len(desde.split('_')) > 1:
497                     desde_data = [desde]
498                     print('Está especificado')
499                 else:
500                     desde_data = []
501                     for sa in base_data:
502                         if desde == sa.split('_')[0]:
503                             desde_data.append(sa)
504                 # Cheking the TO models
505                 if len(hacia.split('_')) > 1:
506                     hacia_data = [hacia]
507                     print('Está especificado')
508                 else:
509                     hacia_data = []
510                     for fa in base_data:
511                         if hacia == fa.split('_')[0]:
512                             hacia_data.append(fa)
513                 # Writing the connections
514                 data_connections = []
515                 for se in desde_data:
516                     for ta in hacia_data:
517                         data_connections.append('connect_many_to_one(
    world,'+str(se)+','+str(ta)+",")
518                 # print(data_connections)
519                 for wa in range(len(data_connections)):
520                     con_run = str(data_connections[wa])
521                     # con_run = 'connect_many_to_one(world,'+str(con[
    i]['FROM'])+','+str(con[i]['TO'])+","
522                     for a in range(len(con[i]['ATTRIBUTES'])):
523                         for k,v in con[i]['ATTRIBUTES'][a].items():
524                             if type(v) == list:
525                                 con_run = con_run+"("
526                                 for t in range(len(v)):
527                                     con_run = con_run+"'%s',"%str(v[t
    ])
528                                 con_run = con_run[:-1] # Eliminar ú
    ltima coma
529                                 con_run = con_run+'),'
530                             else:
```

```
531                                     if con_run[-1] == ',':
532                                         con_run = con_run[:-1] # Eliminar
    última coma
533                                         con_run = con_run+",'%s'"%v
534                                     else:
535                                         con_run = con_run+",'%s'"%v
536                         if con_run[-1] == ',':
537                             con_run = con_run[:-1] # Eliminar última coma
538                         con_run = con_run+')'
539                         # print('conexión many-to-many:',con_run)
540                         eval(con_run)
541                 if con[i]['TYPE'] == 'Direct':
542                     hacia = con[i]['TO']
543                     desde = con[i]['FROM']
544                     # Cheking the FROM model
545                     if len(desde.split('_')) > 1:
546                         desde_data = [desde]
547                         print('Está especificado')
548                     else:
549                         desde_data = []
550                         for sa in base_data:
551                             if desde == sa.split('_')[0]:
552                                 desde_data.append(sa)
553                     # Cheking the TO models
554                     if len(hacia.split('_')) > 1:
555                         hacia_data = [hacia]
556                         print('Está especificado')
557                     else:
558                         hacia_data = []
559                         for fa in base_data:
560                             if hacia == fa.split('_')[0]:
561                                 hacia_data.append(fa)
562                     # Writing the connections
563                     data_connections = []
564                     for se in desde_data:
565                         for ta in hacia_data:
566                             data_connections.append('world.connect('+str(
    se)+','+str(ta)+",")
567                     # print(data_connections)
568                     for wa in range(len(data_connections)):
569                         con_run = str(data_connections[wa])
570                         # con_run = 'world.connect('+str(con[i]['FROM'])
    +','+str(con[i]['TO'])+","
571                         for a in range(len(con[i]['ATTRIBUTES'])):
572                             for k,v in con[i]['ATTRIBUTES'][a].items():
573                                 if type(v) == list:
574                                     con_run = con_run+"("
575                                     for t in range(len(v)):
```

```
576                                         con_run = con_run+"'%s',"%str(v[t
    ])
577                                     con_run = con_run[:-1] # Eliminar ú
    ltima coma
578                                     con_run = con_run+'),'
579                                 else:
580                                     if con_run[-1] == ',':
581                                         con_run = con_run[:-1] # Eliminar
     última coma
582                                         con_run = con_run+",'%s'"%v
583                                     else:
584                                         con_run = con_run+",'%s'"%v
585                         if con_run[-1] == ',':
586                             con_run = con_run[:-1] # Eliminar última coma
587                         con_run = con_run+')'
588                         # print('conexion direct:',con_run)
589                         eval(con_run)
590                 if con[i]['TYPE'] == 'Cyclic':
591                     con_run = 'world.connect('+str(con[i]['FROM'])+','+
    str(con[i]['TO'])+","
592                     for a in range(len(con[i]['ATTRIBUTES'])):
593                         for k,v in con[i]['ATTRIBUTES'][a].items():
594                             if type(v) == list:
595                                 con_run = con_run+"("
596                                 for t in range(len(v)):
597                                     con_run = con_run+"'%s',"%str(v[t])
598                                 con_run = con_run[:-1] # Eliminar última
    coma
599                                 con_run = con_run+'),'
600                             else:
601                                 if con_run[-1] == ',':
602                                     con_run = con_run[:-1] # Eliminar ú
    ltima coma
603                                     con_run = con_run+",'%s'"%v
604                                 else:
605                                     con_run = con_run+",'%s'"%v
606                     if con_run[-1] == ',':
607                         con_run = con_run[:-1] # Eliminar última coma
608                     con_run = con_run+',async_requests=True)'
609                     # print('conexión cyclic:',con_run)
610                     # eval(con_run)
611                 if con[i]['TYPE'] == 'Nodes Many to One':
612                     hacia = con[i]['TO']
613                     desde = con[i]['FROM']
614                     # Cheking the FROM model
615                     if len(desde.split('_')) > 1:
616                         desde_data = [desde]
617                         print('Está especificado')
618                     else:
```

```
619                     desde_data = []
620                     for sa in base_data:
621                         if desde == sa.split('_')[0]:
622                             desde_data.append(sa)
623                 # Cheking the TO models
624                 if len(hacia.split('_')) > 1:
625                     hacia_data = [hacia]
626                     print('Está especificado')
627                 else:
628                     hacia_data = []
629                     for fa in base_data:
630                         if hacia == fa.split('_')[0]:
631                             hacia_data.append(fa)
632                 # Writing the connections
633                 data_connections = []
634                 string_data = []
635                 for se in desde_data:
636                     for ta in hacia_data:
637                         string_data.append('connect_many_to_one(world
    ,'+str(se)+','+str(ta))
638                         data_connections.append('connect_many_to_one(
    world,Nodes'+','+str(ta))
639                 # print(data_connections)
640                 for wa in range(len(data_connections)):
641                     # Nodes = eval("[e for e in %s if e.type in ('
    RefBus, PQBus')]"%con[i]['FROM'])
642                     Nodes = eval("[e for e in %s if e.type in ('
    RefBus, PQBus')]"%str(string_data[wa].split(",")[1]))
643                     # print('los nodes son:', Nodes)
644                     # con_run = 'connect_many_to_one(world,Nodes,'+
    str(con[i]['TO'])
645                     con_run = str(data_connections[wa])
646                     for a in range(len(con[i]['ATTRIBUTES'])):
647                         for k,v in con[i]['ATTRIBUTES'][a].items():
648                             con_run = con_run+",'%s'"%v
649                     con_run = con_run+')'
650                     # print('conexión nodes-many-to-one:',con_run)
651                     eval(con_run)
652             if con[i]['TYPE'] == 'Branches Many to One':
653                 hacia = con[i]['TO']
654                 desde = con[i]['FROM']
655                 # Cheking the FROM model
656                 if len(desde.split('_')) > 1:
657                     desde_data = [desde]
658                     print('Está especificado')
659                 else:
660                     desde_data = []
661                     for sa in base_data:
662                         if desde == sa.split('_')[0]:
```

```
663                             desde_data.append(sa)
664                 # Cheking the TO models
665                 if len(hacia.split('_')) > 1:
666                     hacia_data = [hacia]
667                     print('Está especificado')
668                 else:
669                     hacia_data = []
670                     for fa in base_data:
671                         if hacia == fa.split('_')[0]:
672                             hacia_data.append(fa)
673                 # Writing the connections
674                 data_connections = []
675                 string_data = []
676                 for se in desde_data:
677                     for ta in hacia_data:
678                         string_data.append('connect_many_to_one(world
    ,'+str(se)+','+str(ta)+",")
679                         data_connections.append('connect_many_to_one(
    world,Branches'+','+str(ta))
680                 # print(data_connections)
681                 for wa in range(len(data_connections)):
682                     # Branches = eval("[e for e in %s if e.type in ('
    Transformer', 'Branch')]"%con[i]['FROM'])
683                     Branches = eval("[e for e in %s if e.type in ('
    Transformer', 'Branch')]"%str(string_data[wa].split(",")[1]))
684                     # print('los branches son:',Branches)
685                     # con_run = 'connect_many_to_one(world,Branches
    ,'+str(con[i]['TO'])
686                     con_run = str(data_connections[wa])
687                     for a in range(len(con[i]['ATTRIBUTES'])):
688                         for k,v in con[i]['ATTRIBUTES'][a].items():
689                             con_run = con_run+",'%s'"%v
690                     con_run = con_run+')'
691                     # print('conexión branches-many-to-one:',con_run)
692                     eval(con_run)
693             if con[i]['TYPE'] == 'Nodes Randomly':
694                 hacia = con[i]['TO']
695                 desde = con[i]['FROM']
696                 # Cheking the FROM model
697                 if len(desde.split('_')) > 1:
698                     desde_data = [desde]
699                     print('Está especificado')
700                 else:
701                     desde_data = []
702                     for sa in base_data:
703                         if desde == sa.split('_')[0]:
704                             desde_data.append(sa)
705                 # Cheking the TO models
706                 if len(hacia.split('_')) > 1:
```

```
707                     hacia_data = [hacia]
708                     print('Está especificado')
709                 else:
710                     hacia_data = []
711                     for fa in base_data:
712                         if hacia == fa.split('_')[0]:
713                             hacia_data.append(fa)
714                 # Writing the connections
715                 data_connections = []
716                 string_data = []
717                 for se in desde_data:
718                     for ta in hacia_data:
719                         string_data.append('connect_randomly(world,'+
    str(se)+','+str(ta)+",")
720                         data_connections.append('connect_randomly(
    world,'+str(se)+',Nodes_grid')
721                 # print(data_connections)
722                 for wa in range(len(data_connections)):
723                     # Nodes_grid = eval("[e for e in %s if 'node' in
    e.eid]"%con[i]['TO'])
724                     Nodes_grid = eval("[e for e in %s if 'node' in e.
    eid]"%str(string_data[wa].split(",")[2]))
725                     # print('los nodes-random son:',Nodes_grid)
726                     con_run = str(data_connections[wa])
727                     # con_run = 'connect_randomly(world,'+str(con[i
    ]['FROM'])+',Nodes_grid'
728                     for a in range(len(con[i]['ATTRIBUTES'])):
729                         for k,v in con[i]['ATTRIBUTES'][a].items():
730                             con_run = con_run+",'%s'"%v
731                     con_run = con_run+')'
732                     # print('conexión nodes-randomly:',con_run)
733                     eval(con_run)
734             if con[i]['TYPE'] == 'Building to grid':
735 
736                 hacia = con[i]['TO']
737                 desde = con[i]['FROM']
738                 # Cheking the FROM model
739                 if len(desde.split('_')) > 1:
740                     desde_data = [desde]
741                     print('Está especificado')
742                 else:
743                     desde_data = []
744                     for sa in base_data:
745                         if desde == sa.split('_')[0]:
746                             desde_data.append(sa)
747                 # Cheking the TO models
748                 if len(hacia.split('_')) > 1:
749                     hacia_data = [hacia]
750                     print('Está especificado')
```

```
751                 else:
752                     hacia_data = []
753                     for fa in base_data:
754                         if hacia == fa.split('_')[0]:
755                             hacia_data.append(fa)
756                 # Writing the connections
757                 data_connections = []
758                 # buses_data = []
759                 string_data = []
760                 for se in desde_data:
761                     for ta in hacia_data:
762                         # buses_data.append('connect(world,'+str(se)
    +','+str(ta)+",")
763                         string_data.append('connect(world,'+str(se)+'
    ,'+str(ta)+",")
764                         data_connections.append('connect(world,'+str(
    se)+',Nodes_grid')
765                 # print(data_connections)
766                 for wa in range(len(data_connections)):
767                     # Nodes_grid = eval("[e for e in %s if 'node' in
    e.eid]"%con[i]['TO'])
768                     # Nodes_grid = eval("[e for e in %s if 'node' in
    e.eid]"%str(string_data[wa].split(",")[2]))
769                     # print('los nodes-random son:',Nodes_grid)
770 
771                     buses = eval("filter(lambda e: e.type == 'PQBus',
     %s)"%str(string_data[wa].split(",")[2]))
772                     buses = {b.eid.split('-')[1]: b for b in buses}
773                     # print(buses)
774                     house_data = eval("world.get_data(%s, 'node_id')"
    %str(string_data[wa].split(",")[1]))
775                     houses = eval(string_data[wa].split(",")[1])
776                     for house in houses:
777                         # print('la casa:',house)
778                         node_id = house_data[house]['node_id']
779                         # print('el nodo id:',node_id)
780                         # print('el bus:',buses[node_id])
781                         con_run = "world.connect(house, buses[node_id
    ], ("
782                         for a in range(len(con[i]['ATTRIBUTES'])):
783                             for k,v in con[i]['ATTRIBUTES'][a].items
    ():
784                                 con_run = con_run+"'%s',"%v
785                         con_run = con_run[:-1] # Eliminar última coma
786                         con_run = con_run+'))'
787                         # print('conexión building-to-grid:',con_run)
788                         eval(con_run)
789                         # world.connect(house, buses[node_id], ('
    P_out', 'P'))
```

```
790
791 if __name__ == '__main__':
792     print('central addres:', central_addres)
793     print('ref',ref)
794     if central_addres == ref:
795         main()
796     else:
797         totVal = ""
798         for we in total_info:
799             if we['ip'] == ref:
800                 val = 'python3 '+we['api']+'.py '+we['ip']+':'+we['
    port']+' --remote -t 60 &'
801                 print(val)
802                 totVal = totVal+val
803             else:
804                 print('*********NO ES ACÁ')
805         lastVal = totVal[:-1]
806         subprocess.call(lastVal, shell=True)
```

A.6 Backend Helics simulation

code/opt_scenario_helics.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Sep 16 10:48:16 2022
4
5  @author: Juan
6  """
7  from making_Json import JsonConfig
8  import json
9  import yaml
10 import arrow
11 import gzip
12 import os
13 import copy
14 import numpy as np
15
16 #
       ================================================================
17 # Loading YAML files
18 #
       ================================================================
19 main_Path = os.path.normpath(os.getcwd() + os.sep + os.pardir).
       replace('\\', '/')
```

```
dirs = os.listdir(main_Path)
os.chdir(main_Path)

for file in dirs:
    check = file.split(".")
    if len(check) == 1:
        pass
    else:
        if check[1] == 'yaml':
            if file.split("_")[0] == 'scenario':
                scenario = str(file)
            if file.split("_")[0] == 'Models':
                modelos = str(file)
        else:pass

#=============================================================================

# Opening YAML files
#=============================================================================

with open(scenario) as file:
    data = yaml.load(file, Loader=yaml.FullLoader)

with open(modelos) as file:
    models = yaml.load(file, Loader=yaml.FullLoader)

#=============================================================================

# Removing previous json files
#=============================================================================

current_Path = os.getcwd().replace('\\', '/')
json_Path = current_Path+'/Helics'+'/Jsons/'

dirs = os.listdir(json_Path)
team = []
for file in dirs:
    check = file.split(".")
    if len(check) == 1:
        pass
    else:
        if check[1] == 'json':
            os.remove(str(json_Path)+'/'+str(file))

# Back to main path
```

```
61 current_Path = os.getcwd().replace('\\', '/')
62 main_Path = current_Path+'/Helics'
63 os.chdir(main_Path)
64
65 #
    ================================================================================

66 # Here, I am generating all the JSON file for federates and the
    scenario config
67 #
    ================================================================================

68 config = {}
69 federates = []
70 configFederates = []
71 current_Path = os.getcwd().replace('\\', '/')
72 json_Path = current_Path+'/Jsons'
73
74
75 n = len(data['SCENARIO SCHEMA YAML']['SIMULATORS CONFIGURATION']['
    Simulator'])
76 rueda = data['SCENARIO SCHEMA YAML']['SCENARIO CONFIGURATION']
77 sim = data['SCENARIO SCHEMA YAML']['SIMULATORS CONFIGURATION']['
    Simulator']
78 db_data = data['SCENARIO SCHEMA YAML']['CONNECTIONS']
79 num_sce = 1
80
81 #
    ================================================================================

82 # Knowing the number of scenarios
83 #
    ================================================================================

84 for i in range(len(rueda)):
85     for k,v in rueda[i].items():
86         if k == 'NUMBER':
87             num_sce = int(v)
88
89 #
    ================================================================================

90 # Creando info para la base de datos
91 #
    ================================================================================

92 for i in range(len(sim)):
93     val = sim[i]['API_HELICS'].split("_")
94     if val[0] == 'hdf5':
```

```
95          db_name = sim[i]['MODELS'][0]['MODEL_NAME']
96          # print(db_name)
97          break
98      else:
99          db_name = 'Nothing'
100
101 info = []
102 # print(type(db_data))
103 if db_name == 'Nothing':
104     pass
105 else:
106     if db_data is None:
107         pass
108     else:
109         for i in range(len(db_data)):
110             if db_data[i]['TO'] == db_name:
111                 for l in range(len(db_data[i]['ATTRIBUTES'])):
112                     for k,v in db_data[i]['ATTRIBUTES'][l].items():
113                         x = {db_data[i]['FROM']:v}
114                         # x = {db_data[i]['FROM']}
115                         info.append(x)
116             else:
117                 pass
118
119 #
    ============================================================================

120 # Generating info for publications and subscriptions
121 #
    ============================================================================


122
123 pub_sub_info = []
124 for ct in range(len(sim)):
125     for mn in range(int(sim[ct]['MODELS'][0]['NUMBER'])):
126         pub_sub_info.append(str(sim[ct]['MODELS'][0]['MODEL_NAME']+'_
    '+str(mn)))
127
128
129 #
    ============================================================================

130 # Getting info about IP addres and port
131 #
    ============================================================================

132 broker_host = ""
133 broker_port = ""
134 for q in range(len(rueda)):
```

```
135     for qy,vu in rueda[q].items():
136         if qy == 'BROKER_HOST':
137             broker_host = vu
138         if qy == 'BROKER_PORT':
139             broker_port = vu
140 
141 #    ============================================================================
142 # Starting the config information
143 #    ============================================================================
144 
145 for i in range(n):
146     if broker_host == 'localhost':
147         Simulator = {"Simulator_Name": "",
148                 "Model_Name": "",
149                 "key":"",
150                 "Federate_host": "",
151                 "Model_instance":{"nameFederate": "",
152                                   "namePubs": {},
153                                   "nameSubs": {},
154                                   "startDate":"",
155                                   "amount":1,
156                                   "num_fed":1,
157                                   "num_inst":1},
158                 "Inputs":{}}
159     else:
160         Simulator = {"Simulator_Name": "",
161             "Model_Name": "",
162             "broker_address": "tcp://"+str(broker_host)+":"+str(
    broker_port),
163             "key":"",
164             "Federate_host": "",
165             "Model_instance":{"nameFederate": "",
166                               "namePubs": {},
167                               "nameSubs": {},
168                               "startDate":"",
169                               "amount":1,
170                               "num_fed":1,
171                               "num_inst":1},
172             "Inputs":{}}
173 
174     for a in range(len(sim[i]['RUN PROCESS'])):
175         for k,v in sim[i]['RUN PROCESS'][a].items():
176             if k == 'MODE_RUN_HELICS':
177                 if v == 'python':
```

```
178                     Simulator['Simulator_Name'] = sim[i]['API_HELICS']+'.py'
179             if k == 'HOSTPORT':
180                 Simulator['Federate_host'] = v
181
182     Simulator['Model_Name'] = sim[i]['SIMULATOR_NAME']
183
184     # Adding model name and inputs
185
186     for m in range(len(models['MODELS'])):
187         for t in range(len(sim[i]['MODELS'])):
188             if models['MODELS'][m]['MODEL_NAME'] == sim[i]['MODELS'][t]['MODEL_NAME']:
189                 midx = t
190                 Simulator['Model_instance']['nameFederate'] = sim[i]['MODELS'][t]['MODEL_NAME']
191                 Simulator['Model_instance']['num_fed'] = sim[i]['MODELS'][t]['NUMBER'] # Taking how many federates I want to create
192                 if not models['MODELS'][m]['INPUT']:
193                     pass
194                 else:
195                     for r in range(len(models['MODELS'][m]['INPUT'])):
196                         for key, value in models['MODELS'][m]['INPUT'][r].items():
197                             Simulator['Inputs'][key] = value
198
199     # Adding parameters
200
201     if not sim[i]['PARAMETERS']:
202         pass
203     else:
204         for p in range(len(sim[i]['PARAMETERS'])):
205             for k,v in sim[i]['PARAMETERS'][p].items():
206                 if k == 'timeAdvance' and v != 0:
207                     Simulator['Model_instance'][k] = v
208                 elif k == 'stepTime' and v != 0:
209                     Simulator['Model_instance'][k] = v
210                 elif k == 'sim_start':
211                     Simulator['Model_instance'][k] = v
212                 elif k == 'datafile':
213                     Simulator['Model_instance'][k] = v
214                 else:
215                     pass
216
217     # """Begining new version"""
218     if not data['SCENARIO SCHEMA YAML']['CONNECTIONS']:
219         for s in range(Simulator['Model_instance']['num_fed']):
220             configFederates.append(Simulator)
```

```
221                 # print('Hola')
222                 pass
223         else:
224             con = data['SCENARIO SCHEMA YAML']['CONNECTIONS']
225             from_base = ""
226             to_base = ""
227             var_Federates = False
228             newSimulator = copy.deepcopy(Simulator)
229             for c in range(len(con)):
230                 # from_base = ""
231                 # to_base = ""
232                 if con[c]['FROM'].split('_')[0] == sim[i]['MODELS'][midx
    ]['MODEL_NAME']:
233                     if len(con[c]['FROM'].split('_')) > 1:
234                         # print('primero-----', con[c]['FROM'])
235                         from_base = str(con[c]['FROM'].split('_')[0])
236                         if int(con[c]['FROM'].split('_')[1]) == int(s):
237                             # print('primero-----', s)
238                             if from_base == 'ResidentialLoads':
239                                 # print(len(con[c]['ATTRIBUTES']))
240                                 if len(con[c]['ATTRIBUTES']) == 1:
241                                     pass
242                                 else:
243                                     con[c]['ATTRIBUTES'] = [con[c]['
    ATTRIBUTES'][0]]
244                                 nuevo_data = newSimulator['Inputs']
245                                 pf = nuevo_data['profile_file']
246                                 archivo = gzip.open(pf, 'rt')
247                                 assert next(archivo).startswith('#
    meta')
248                                 meta = json.loads(next(archivo))
249                                 assert next(archivo).startswith('#
    id_list')
250                                 id_list_lines = []
251                                 for line in archivo:
252                                     if line.startswith('# attrs'):
253                                         break
254                                     id_list_lines.append(line)
255                                 id_lists = json.loads(''.join(
    id_list_lines))
256                                 houses_quantity = len(list(id_lists.
    values())[0])
257                                 con[c]['TOTAL_PUB'] = houses_quantity
258                                 con[c]['TOTAL_SUB'] = houses_quantity
259                             if from_base == 'PV':
260                                 for w in range(len(models['MODELS'])):
261                                     if models['MODELS'][w]['MODEL_NAME']
    == con[c]['FROM']:
```

```
262                                         con[c]['TOTAL_PUB'] = models['
    MODELS'][w]['PARAMETERS'][0]['NUMBER']
263                                         con[c]['TOTAL_SUB'] = models['
    MODELS'][w]['PARAMETERS'][0]['NUMBER']
264                                 for a in range(len(con[c]['ATTRIBUTES'])):
265                                     for k,v in  con[c]['ATTRIBUTES'][a].items
    ():
266                                         if k == 'ATTR_FROM' or k == 'ATTR_TO'
     or k == 'ATTR':
267                                             # print('este es el valor de V:',
    v)
268                                             newSimulator['Model_instance']['
    namePubs'][str(con[c]['FROM'])+str(v)] = con[c]['TOTAL_PUB']
269                                 # print('******',Simulator['Model_instance
    ']['namePubs'])
270                                 var_Federates = True
271                                 # configFederates.append(newSimulator)
272                             else:
273                                 # print('.....',s)
274                                 var_Federates = False
275                                 # configFederates.append(Simulator)
276                                 # print('******',Simulator['Model_instance
    ']['namePubs'])
277                         else:
278                             # print('segundo --------')
279                             for mn in range(len(pub_sub_info)):
280                                 if pub_sub_info[mn].split('_')[0] == con[c]['
    FROM']:
281                                     from_base = pub_sub_info[mn].split('_')
    [0]
282                                     if from_base == 'ResidentialLoads':
283                                         if len(con[c]['ATTRIBUTES']) == 1:
284                                             pass
285                                         else:
286                                             con[c]['ATTRIBUTES'] = [con[c]['
    ATTRIBUTES'][0]]
287                                         nuevo_data = newSimulator['Inputs']
288                                         pf = nuevo_data['profile_file']
289                                         archivo = gzip.open(pf, 'rt')
290                                         assert next(archivo).startswith('#
    meta')
291                                         meta = json.loads(next(archivo))
292                                         assert next(archivo).startswith('#
    id_list')
293                                         id_list_lines = []
294                                         for line in archivo:
295                                             if line.startswith('# attrs'):
296                                                 break
297                                             id_list_lines.append(line)
```

```
298                                     id_lists = json.loads(''.join(
    id_list_lines))
299                                     houses_quantity = len(list(id_lists.
    values())[0])
300                                     con[c]['TOTAL_PUB'] = houses_quantity
301                                     con[c]['TOTAL_SUB'] = houses_quantity
302                                     # print('estas son las total pub',con
    [c]['TOTAL_PUB'] )
303                                 if from_base == 'PV':
304                                     for w in range(len(models['MODELS'])
    ):
305                                         if models['MODELS'][w]['
    MODEL_NAME'] == con[c]['FROM']:
306                                             con[c]['TOTAL_PUB'] = models[
    'MODELS'][w]['PARAMETERS'][0]['NUMBER']
307                                             con[c]['TOTAL_SUB'] = models[
    'MODELS'][w]['PARAMETERS'][0]['NUMBER']
308                                 for a in range(len(con[c]['ATTRIBUTES']))
    :
309                                     for k,v in  con[c]['ATTRIBUTES'][a].
    items():
310                                         if k == 'ATTR_FROM' or k == '
    ATTR_TO' or k == 'ATTR':
311                                             newSimulator['Model_instance'
    ]['namePubs'][str(pub_sub_info[mn])+str(v)] = con[c]['TOTAL_PUB']
312                         # configFederates.append(Simulator)
313             if con[c]['TO'].split('_')[0] == sim[i]['MODELS'][midx]['
    MODEL_NAME']:
314                 if len(con[c]['TO'].split('_')) > 1:
315                     to_base = str(con[c]['TO'].split('_')[0])
316                     if int(con[c]['TO'].split('_')[1]) == int(s):
317                         if con[c]['FROM'] == 'PV':
318                             for w in range(len(models['MODELS'])):
319                                 if models['MODELS'][w]['MODEL_NAME']
    == con[c]['FROM']:
320                                     con[c]['TOTAL_PUB'] = models['
    MODELS'][w]['PARAMETERS'][0]['NUMBER']
321                                     con[c]['TOTAL_SUB'] = models['
    MODELS'][w]['PARAMETERS'][0]['NUMBER']
322                         for b in range(len(con[c]['ATTRIBUTES'])):
323                             for k,v in  con[c]['ATTRIBUTES'][b].items
    ():
324                                 if k == 'ATTR_FROM' or k == 'ATTR_TO'
     or k == 'ATTR':
325                                     newSimulator['Model_instance']['
    nameSubs'][str(mn)+str(v)] = con[c]['TOTAL_SUB']
326                 else:
327                     # print('segundo --------')
328                     if len(con[c]['FROM'].split('_')) > 1:
```

```
329                         for tn in range(len(pub_sub_info)):
330                             if pub_sub_info[tn] == con[c]['FROM']:
331                                 if con[c]['FROM'] == 'PV':
332                                     for w in range(len(models['MODELS
    ']))):
333                                         if models['MODELS'][w]['
    MODEL_NAME'] == con[c]['FROM']:
334                                             con[c]['TOTAL_PUB'] =
    models['MODELS'][w]['PARAMETERS'][0]['NUMBER']
335                                             con[c]['TOTAL_SUB'] =
    models['MODELS'][w]['PARAMETERS'][0]['NUMBER']
336                                 for b in range(len(con[c]['ATTRIBUTES
    ']))):
337                                     for k,v in  con[c]['ATTRIBUTES'][
    b].items():
338                                         if k == 'ATTR_FROM' or k == '
    ATTR_TO' or k == 'ATTR':
339                                             newSimulator['
    Model_instance']['nameSubs'][str(con[c]['FROM'])+str(v)] = con[c][
    'TOTAL_SUB']
340                     else:
341                         for tn in range(len(pub_sub_info)):
342                             if pub_sub_info[tn].split('_')[0] == con[
    c]['FROM']:
343                                 if con[c]['FROM'] == 'PV':
344                                     for w in range(len(models['MODELS
    ']))):
345                                         if models['MODELS'][w]['
    MODEL_NAME'] == con[c]['FROM']:
346                                             con[c]['TOTAL_PUB'] =
    models['MODELS'][w]['PARAMETERS'][0]['NUMBER']
347                                             con[c]['TOTAL_SUB'] =
    models['MODELS'][w]['PARAMETERS'][0]['NUMBER']
348                                 for b in range(len(con[c]['ATTRIBUTES
    ']))):
349                                     for k,v in  con[c]['ATTRIBUTES'][
    b].items():
350                                         if k == 'ATTR_FROM' or k == '
    ATTR_TO' or k == 'ATTR':
351                                             newSimulator['
    Model_instance']['nameSubs'][str(pub_sub_info[tn])+str(v)] = con[c
    ]['TOTAL_SUB']
352
353             if con[c]['TO'] == 'Database' and sim[i]['MODELS'][midx][
    'MODEL_NAME'] == 'DB':
354                 for d in range(len(con[c]['ATTRIBUTES'])):
355                     for k,v in  con[c]['ATTRIBUTES'][d].items():
356                         if k == 'ATTR_FROM' or k == 'ATTR_TO' or k ==
     'ATTR':
```

```
357                                 newSimulator['Model_instance']['nameSubs'
    ][str(con[c]['FROM'])+str(v)] = con[c]['TOTAL_SUB']
358
359             configFederates.append(newSimulator)
360           # """End new version"""
361 newConfigFederates = []
362
363 for i in configFederates:
364     for k in range (int(i['Model_instance']['num_fed'])):
365         newConfigFederates.append(copy.deepcopy(i))
366
367 # Renaming confiFederates
368
369 sim_name_dos = []
370 # copy_new_Federates = copy.deepcopy(new_Federates)
371 for g in range(len(newConfigFederates)):
372     name_dos = newConfigFederates[g]['Model_Name']
373     n_name_dos = 0
374     for s in sim_name_dos:
375         if s.split("_")[0] == name_dos:
376             n_name_dos += 1
377     sim_nombre_dos = str(name_dos)+'_'+str(n_name_dos)
378     sim_name_dos.append(sim_nombre_dos)
379     newConfigFederates[g]['Model_name'] = sim_nombre_dos
380
381 # Checking numPubs
382 new_info_federates = copy.deepcopy(newConfigFederates)
383 for nt in range(len(newConfigFederates)):
384     var_elim = []
385     myvar1 = newConfigFederates[nt]['Model_name'].split('_')[1]
386     myvar2 = newConfigFederates[nt]['Model_instance']['nameFederate']
387     myvar3 = myvar2+'_'+myvar1
388     for k,v in newConfigFederates[nt]['Model_instance']['namePubs'].
    items():
389         if len(k.split(myvar3)) > 1 and k.split(myvar3)[0] == '' and
    k.split(myvar3)[1][0].isnumeric() != True:
390             print(k)
391         else:
392             var_elim.append(k)
393     for kl in range(len(var_elim)):
394         del newConfigFederates[nt]['Model_instance']['namePubs'][
    var_elim[kl]]
395
396 # Making JSON config for each federate
397 names = []
398 for f in range(len(newConfigFederates)):
399     jsonfile = newConfigFederates[f]
400     # print()
401     # jsonfile['Model_name'] = sim_name_dos[f]
```

```
402     # print(sim_name_dos[f])
403     gen = JsonConfig(jsonfile)
404     jsonname = gen.run()
405     names.append(jsonname)
406 
407 # Taking number of days from YAML
408 days = 0
409 for q in range(len(rueda)):
410     for k,v in rueda[q].items():
411         if k == 'DAYS':
412             days = v
413         else:
414             pass
415 # """Writen Scenario info"""
416 # "exec": "helics_broker -f "+str(len(data['SCENARIO SCHEMA YAML']['
    SIMULATORS CONFIGURATION']['Simulator']))+" --key="""+" --loglevel
    =1",
417 if broker_port == "":
418     scenario = {"directory": "./",
419             "exec": "helics_broker -f "+str(len(newConfigFederates))+
    " --all --loglevel=TRACE",
420             "host": data['SCENARIO SCHEMA YAML']['SCENARIO
    CONFIGURATION'][3]['BROKER_HOST'],
421             "name": data['SCENARIO SCHEMA YAML']['SCENARIO
    CONFIGURATION'][5]['BROKER_NAME']}
422     federates.append(scenario)
423 else:
424     scenario = {"directory": "./",
425                 "exec": "helics_broker -f "+str(len(
    newConfigFederates))+" --all --loglevel=TRACE --port="+str(data['
    SCENARIO SCHEMA YAML']['SCENARIO CONFIGURATION'][4]['BROKER_PORT'
    ]),
426                 "host": data['SCENARIO SCHEMA YAML']['SCENARIO
    CONFIGURATION'][3]['BROKER_HOST'],
427                 "name": data['SCENARIO SCHEMA YAML']['SCENARIO
    CONFIGURATION'][5]['BROKER_NAME']}
428     federates.append(scenario)
429 
430 # """Writen Simulators info"""
431 for i in range(len(newConfigFederates)):
432     ## Espacio para insertar pasar META info
433     for n in range(len(sim)):
434         name_model = ""
435         if newConfigFederates[i]['Model_instance']['nameFederate'] ==
     sim[n]['MODELS'][0]['MODEL_NAME']:
436             # if newConfigFederates[i]['Model_Name'] == sim[n]['
    SIMULATOR_NAME']:
437             name_model = sim[n]['MODELS'][0]['MODEL_NAME']
438             for m in range(len(models['MODELS'])):
```

```
439                 if name_model == models['MODELS'][m]['MODEL_NAME']:
440                     for l in range (len(models['MODELS'][m]['Model
    instance'])):
441                         for k,v in models['MODELS'][m]['Model
    instance'][l].items():
442                             if k == 'ATTRS':
443                                 meta_attr = v # Tomando los
    attributos del simulador
444                     meta_parm = []
445                     parm_input = {}
446                     for ke,va in models['MODELS'][m]['PARAMETERS'
    ][0].items():
447                         if ke == 'PARAMS_SET_NAME' or ke == 'NUMBER':
448                             pass
449                         else:
450                             meta_parm.append(ke) # Tomando los pará
    metros del simulador
451                             parm_input[ke] = va
452     parm_input = str(parm_input).replace(" ", "").replace("'", '*')
453     # print('Meta es:', meta_parm)
454     # print('Otro meta es:',parm_input)
455     ## Espacio para insertar pasar META info
456     if not meta_parm:
457         if len(newConfigFederates[i]['Inputs']) != 0:
458             for k,v in newConfigFederates[i]['Model_instance'].items
    ():
459                 if k != 'timeAdvance':
460                     simulator = {"directory": "./",
461                             "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+'days:'+str(days)+" "+'name:'+str(names[i
    ]),
462                             "host": newConfigFederates[i]['
    Federate_host'],
463                             "name": sim_name_dos[i]}
464                     for key, value in newConfigFederates[i]['Inputs'
    ].items():
465                         if key == 'info' and value == 'yes':
466                             x =  simulator['exec']
467                             simulator['exec'] = x+" "+str(key)+":"+
    str(info).replace(" ", "")
468                         else:
469                             x =  simulator['exec']
470                             simulator['exec'] = x+" "+str(key)+":"+
    str(value)
471                     # federates.append(simulator)
472                 else:
473                     simulator = {"directory": "./",
```

```
474                             "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+str(newConfigFederates[i]['Model_instance
    ']['timeAdvance'])+" "+'days:'+str(days)+" "+'name:'+str(names[i])
    ,
475                             "host": newConfigFederates[i]['
    Federate_host'],
476                             "name": sim_name_dos[i]}
477                     for key, value in newConfigFederates[i]['Inputs'
    ].items():
478                         if key == 'info' and value == 'yes':
479                             x =  simulator['exec']
480                             simulator['exec'] = x+" "+str(key)+":"+
    str(info).replace(" ", "")
481                         else:
482                             x =  simulator['exec']
483                             simulator['exec'] = x+" "+str(key)+":"+
    str(value)
484             federates.append(simulator)
485         else:
486             simulator = {"directory": "./",
487                             "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+'days:'+str(days)+" "+'name:'+str(names[i
    ]),
488                             "host": newConfigFederates[i]['
    Federate_host'],
489                             "name": sim_name_dos[i]}
490             for k,v in newConfigFederates[i]['Model_instance'].items
    ():
491                 if k != 'nameFederate' and k != 'namePubs' and k != '
    nameSubs' and k != 'startDate' and k != 'amount' and k != '
    metadata' and k != 'stepTime' and k != 'num_fed':
492                     if k == 'timeAdvance':
493                         x =  simulator['exec']
494                         simulator['exec'] = x+" "+str(v)
495                     else:
496                         x =  simulator['exec']
497                         simulator['exec'] = x+" "+str(k)+":"+str(v)
498                 else:
499                     pass
500                     # simulator = {"directory": "./",
501                     #         "exec": "python -u "+
    newConfigFederates[i]['Simulator_Name']+" "+str(newConfigFederates
    [i]['Model_instance']['timeAdvance'])+" "+'days:'+str(days)+" "+'
    name:'+str(names[i]),
502                     #         "host": newConfigFederates[i]['
    Federate_host'],
503                     #         "name": newConfigFederates[i]['
    Model_Name']}
504             federates.append(simulator)
```

```
505     else:
506         if len(newConfigFederates[i]['Inputs']) != 0:
507             for k,v in newConfigFederates[i]['Model_instance'].items
    ():
508                 if k != 'timeAdvance':
509                     simulator = {"directory": "./",
510                                  "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+'days:'+str(days)+" "+'name:'+str(names[i
    ])+" "+'meta_attr:'+str(meta_attr).replace(" ", "")+" "+'meta_parm
    :'+str(meta_parm).replace(" ", "")+" "+'parm_input:'+parm_input,
511                                  "host": newConfigFederates[i]['
    Federate_host'],
512                                  "name": sim_name_dos[i]}
513                     for key, value in newConfigFederates[i]['Inputs'
    ].items():
514                         if key == 'info' and value == 'yes':
515                             x =  simulator['exec']
516                             simulator['exec'] = x+" "+str(key)+":"+
    str(info).replace(" ", "")
517                         else:
518                             x =  simulator['exec']
519                             simulator['exec'] = x+" "+str(key)+":"+
    str(value)
520                     # federates.append(simulator)
521                 else:
522                     simulator = {"directory": "./",
523                                  "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+str(newConfigFederates[i]['Model_instance
    ']['timeAdvance'])+" "+'days:'+str(days)+" "+'name:'+str(names[i])
    +" "+'meta_attr:'+str(meta_attr).replace(" ", "")+" "+'meta_parm:'
    +str(meta_parm).replace(" ", "")+" "+'parm_input:'+parm_input,
524                                  "host": newConfigFederates[i]['
    Federate_host'],
525                                  "name": sim_name_dos[i]}
526                     for key, value in newConfigFederates[i]['Inputs'
    ].items():
527                         if key == 'info' and value == 'yes':
528                             x =  simulator['exec']
529                             simulator['exec'] = x+" "+str(key)+":"+
    str(info).replace(" ", "")
530                         else:
531                             x =  simulator['exec']
532                             simulator['exec'] = x+" "+str(key)+":"+
    str(value)
533             federates.append(simulator)
534         else:
535             simulator = {"directory": "./",
```

```
536                                 "exec": "python -u "+ newConfigFederates[
    i]['Simulator_Name']+" "+'days:'+str(days)+" "+'name:'+str(names[i
    ]),
537                                 "host": newConfigFederates[i]['
    Federate_host'],
538                                 "name": sim_name_dos[i]}
539                 for k,v in newConfigFederates[i]['Model_instance'].items
    ():
540                     if k != 'nameFederate' and k != 'namePubs' and k != '
    nameSubs' and k != 'startDate' and k != 'amount' and k != '
    metadata' and k != 'stepTime' and k != 'num_fed':
541                         if k == 'timeAdvance':
542                             x =  simulator['exec']
543                             simulator['exec'] = x+" "+str(v)
544                         else:
545                             x =  simulator['exec']
546                             simulator['exec'] = x+" "+str(k)+":"+str(v)
547                     else:
548                         pass
549                         # simulator = {"directory": "./",
550                         #         "exec": "python -u "+
    newConfigFederates[i]['Simulator_Name']+" "+str(newConfigFederates
    [i]['Model_instance']['timeAdvance'])+" "+'days:'+str(days)+" "+'
    name:'+str(names[i]),
551                         #         "host": newConfigFederates[i]['
    Federate_host'],
552                         #         "name": newConfigFederates[i]['
    Model_Name']}
553                 federates.append(simulator)
554
555
556 """Generating config JSON file"""
557 config["federates"] = federates
558 config["name"] = data['SCENARIO SCHEMA YAML']['SCENARIO CONFIGURATION
    ']['SCENARIO_NAME']
559
560 """Saving JSON file"""
561 for i in range(int(num_sce)):
562     with open('config_'+str(i)+'.json', 'w') as fp:
563         json.dump(config, fp, indent=1)
```

Bibliography

[1] V Cagri Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. «A survey on smart grid potential applications and communication requirements». In: *IEEE Transactions on industrial informatics* 9.1 (2012), pp. 28–42 (cit. on p. 5).

[2] Carlos Andrés Macana, Nicanor Quijano, and Eduardo Mojica-Nava. «A survey on cyber physical energy systems and their applications on smart grids». In: *2011 IEEE PES conference on innovative smart grid technologies Latin America (ISGT LA)*. IEEE. 2011, pp. 1–7 (cit. on p. 5).

[3] RICARDO M CZEKSTER. «Analysis of selected frameworks in Smart Grid co-simulation». In: () (cit. on pp. 5, 6).

[4] Cornelius Steinbrink et al. «Cpes testing with mosaik: Co-simulation planning, execution and analysis». In: *Applied Sciences* 9.5 (2019), p. 923 (cit. on pp. 5, 7).

[5] Ricardo M Czekster. «Tools for modelling and simulating the Smart Grid». In: *arXiv preprint arXiv:2011.07968* (2020) (cit. on pp. 5, 6).

[6] Kevin Mets, Juan Aparicio Ojea, and Chris Develder. «Combining power and communication network simulation for cost-effective smart grid analysis». In: *IEEE Communications Surveys & Tutorials* 16.3 (2014), pp. 1771–1796 (cit. on p. 5).

[7] Peter Palensky, Edmund Widl, and Atiyah Elsheikh. «Simulating cyber-physical energy systems: Challenges, tools and methods». In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.3 (2013), pp. 318–326 (cit. on p. 6).

[8] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. «Co-simulation: a survey». In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–33 (cit. on p. 6).

[9] Cornelius Steinbrink, Arjen A van der Meer, Milos Cvetkovic, Davood Babazadeh, Sebastian Rohjans, Peter Palensky, and Sebastian Lehnhoff. «Smart grid co-simulation with MOSAIK and HLA: a comparison study». In: *Computer Science-Research and Development* 33.1 (2018), pp. 135–143 (cit. on pp. 6, 16, 17).

[10] Peter Palensky, Edmund Widl, Matthias Stifter, and Atiyah Elsheikh. «Modeling intelligent energy systems: Co-simulation platform for validating flexible-demand EV charging management». In: *IEEE Transactions on Smart Grid* 4.4 (2013), pp. 1939–1947 (cit. on p. 6).

[11] Brian M Kelley, Philip Top, Steven G Smith, Carol S Woodward, and Liang Min. «A federated simulation toolkit for electric power grid and communication network co-simulation». In: *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE. 2015, pp. 1–6 (cit. on p. 6).

[12] Claudio David López, Miloš Cvetković, Arjen van der Meer, and Peter Palensky. «Co-simulation of Intelligent Power Systems». In: *Intelligent Integrated Energy Systems*. Springer, 2019, pp. 99–119 (cit. on p. 6).

[13] Peter Palensky, Arjen van der Meer, Claudio Lopez, Arun Joseph, and Kaikai Pan. «Applied cosimulation of intelligent power systems: Implementing hybrid simulators for complex power systems». In: *IEEE Industrial Electronics Magazine* 11.2 (2017), pp. 6–21 (cit. on p. 6).

[14] Mehmet Hazar Cintuglu, Osama A Mohammed, Kemal Akkaya, and A Selcuk Uluagac. «A survey on smart grid cyber-physical system testbeds». In: *IEEE Communications Surveys & Tutorials* 19.1 (2016), pp. 446–464 (cit. on p. 6).

[15] Cornelius Steinbrink, Christian Köhler, Marius Siemonsmeier, and Thorsten van Ellen. «Lessons learned from CPES co-simulation with distributed, heterogeneous systems». In: *Energy Informatics* 1.1 (2018), pp. 327–335 (cit. on p. 6).

[16] Mike Vogt, Frank Marten, and Martin Braun. «A survey and statistical analysis of smart grid co-simulations». In: *Applied energy* 222 (2018), pp. 67–78 (cit. on p. 6).

[17] Steffen Schütte, Stefan Scherfke, and Martin Tröschel. «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. 2011, pp. 55–60 (cit. on pp. 6, 9).

[18] Steffen Schütte, Stefan Scherfke, and Michael Sonnenschein. «Mosaik-smart grid simulation api». In: *Proceedings of SMARTGREENS* (2012), pp. 14–24 (cit. on p. 6).

[19] Sebastian Rohjans, Sebastian Lehnhoff, Steffen Schütte, Stefan Scherfke, and Shahid Hussain. «mosaik-A modular platform for the evaluation of agent-based Smart Grid control». In: *IEEE PES ISGT Europe 2013*. IEEE. 2013, pp. 1–5 (cit. on p. 6).

[20] Kunpeng Wang, Peer-Olaf Siebers, and Darren Robinson. «Towards generalized co-simulation of urban energy systems». In: *Procedia engineering* 198 (2017), pp. 366–374 (cit. on p. 7).

[21] Luca Barbierato, Abouzar Estebsari, Lorenzo Bottaccioli, Enrico Macii, and Edoardo Patti. «A Distributed Multimodel Cosimulation Platform to Assess General Purpose Services in Smart Grids». In: *IEEE Transactions on Industry Applications* 56.5 (2020), pp. 5613–5624 (cit. on p. 7).

[22] Nakisa Farrokhseresht, Arjen A van der Meer, José Rueda Torres, and Mart AMM van der Meijden. «MOSAIK and FMI-Based Co-Simulation Applied to Transient Stability Analysis of Grid-Forming Converter Modulated Wind Power Plants». In: *Applied Sciences* 11.5 (2021), p. 2410 (cit. on p. 7).

[23] Bryan Palmintier, Dheepak Krishnamurthy, Philip Top, Steve Smith, Jeff Daily, and Jason Fuller. «Design of the HELICS high-performance transmission–distribution-communication-market co-simulation framework». In: *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE. 2017, pp. 1–6 (cit. on pp. 8, 13–15).

[24] Welin Zhong, Muyang Liu, and Federico Milano. «A co-simulation framework for power systems and communication networks». In: *2019 IEEE Milan PowerTech*. IEEE. 2019, pp. 1–6 (cit. on p. 8).

[25] Tan Duy Le, Adnan Anwar, Razvan Beuran, and Seng W Loke. «Smart Grid Co-Simulation Tools: Review and Cybersecurity Case Study». In: *2019 7th International Conference on Smart Grid (icSmartGrid)*. IEEE. 2019, pp. 39–45 (cit. on p. 8).

[26] Tan Duy Le, Adnan Anwar, Seng W Loke, Razvan Beuran, and Yasuo Tan. «GridAttackSim: A Cyber Attack Simulation Framework for Smart Grids». In: *Electronics* 9.8 (2020), p. 1218 (cit. on p. 8).

[27] Jianhua Zhang, Jeff Daily, Ryan A Mast, Bryan Palmintier, Dheepak Krishnamurthy, Tarek Elgindy, Anthony Florita, and Bri-Mathias Hodge. «Development of HELICS-based High-Performance Cyber-Physical Co-simulation Framework for Distributed Energy Resources Applications». In: *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE. 2020, pp. 1–5 (cit. on pp. 8, 13–15).

[28] Alok Kumar Bharati and Venkataramana Ajjarapu. «A Scalable Multi-Timescale T&D Co-Simulation Framework using HELICS». In: *2021 IEEE Texas Power and Energy Conference (TPEC)*. IEEE. 2021, pp. 1–6 (cit. on p. 8).

[29] Dylan Cutler, Ted Kwasnik, Sivasathya Balamurugan, Tarek Elgindy, Siddharth Swaminathan, Jeff Maguire, and Dane Christensen. «Co-simulation of transactive energy markets: A framework for market testing and evaluation». In: *International Journal of Electrical Power & Energy Systems* 128 (2021), p. 106664 (cit. on p. 8).

[30] Steffen Schütte. «Simulation model composition for the large-scale analysis of smart grid control mechanisms». PhD thesis. Universität Oldenburg, 2013 (cit. on p. 10).

[31] Daniele Salvatore Schiera, Luca Barbierato, Andrea Lanzini, Romano Borchiellini, Enrico Pons, Ettore Bompard, Edoardo Patti, Enrico Macii, and Lorenzo Bottaccioli. «A distributed multimodel platform to cosimulate multienergy systems in smart buildings». In: *IEEE Transactions on Industry Applications* 57.5 (2021), pp. 4428–4440 (cit. on pp. 23, 24).