Master thesis : Co-simulation platform for cyber-physical transmission systems

Auteur : Godfraind, Adrien
Promoteur(s) : Ernst, Damien
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "electric power and energy systems"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11673

University of Liège
School of Engineering and Computer Science

Co-simulation platform for cyber-physical transmission systems

Master's thesis carried out by

Godfraind Adrien

to obtain the degree of

Master of Science in Electrical Engineering

Academic Supervisor
Damien Ernst (ULiège)

Internship Supervisors
Quentin Gemine
(BlackLight Analytics)
Yves Vanaubel
(BlackLight Analytics)

Academic year 2020-2021

Abstract

In recent years, the energy sector has been in transition and this is particularly visible in the electricity grid. More and more elements of the grid are smart and connected, leading the community to talk about smart grids or cyber-physical power systems. These systems require research and development of new protection and control algorithms to plan and operate the power system and protect it against the multitude of new threats that emerge with the addition of the cyber layer to the physical grid.
This is why the fundamental research project "Cyber-Physical Risk of the bulk Electric Energy Supply System" (CYPRESS) has been launched. The objective of this project is to develop new knowledge, methods and tools needed to guarantee the security of supply through the electricity transmission system.
This Master's thesis has been produced along the lines of the original objectives of this project. These consist, among other things, in selecting a platform adapted to the co-simulation of cyber-physical power systems. First, an in-depth research of the state of the art on cyber-physical power system co-simulation platforms is presented. Through the various stages of this state of the art, choices are made regarding the most appropriate techniques and tools to be used in the project. Then, these choices of existing tools and methods are used in order to realize a practical implementation test case of cyber-physical power system co-simulation. The steps of this practical implementation are described and the results of a test scenario are provided as proof of concept. Finally, research on improving the platform for future developments is presented in the conclusion of the paper, with a focus on making the co-simulation platform more generic.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

For a number of years, the energy sector has been in full transition and this is particularly noticeable in the electricity network. More and more elements of the grid are intelligent and connected, which leads the community to talk about smart grids or cyber-physical power systems (CPPS) [15],[16]. We are seeing more and more distributed generation, decentralized control, agents in the network and in the market, intelligent loads and buildings, autonomous software, etc. being part of the network and its management [17],[5]. All these are enabled and empowered by information and communication technology (ICT), so that the latter is more and more present in electrical systems [5]. From generation to consumption, passing through transmission and distribution, there is clearly an increasingly tight connection between power systems and ICT networks.

This growing interconnection calls for the research and development of new protection and control algorithms [18] to plan and operate the power system, optimize its use or protect it against the multitude of new threats that are emerging with the addition of the cyber layer to the physical network. Indeed, while the ICT presents an enhancement that enables, among other things, complex controls in smart grids, it also introduces additional complexity, new sources of failure and security threats [5].

These cyber threats and disruptions may be intentional or unintentional. Indeed, a lone attacker or a terrorist organisation may intentionally (try to) exploit vulnerabilities introduced by cyber systems to affect the operation of the power network, in this case we speak of cyber-attacks. But unintentional failures can also occur due to software bugs or errors in the configuration of software and networks for example [19].

It is important to note that threats to (cyber-physical) power systems may have quite serious and even dangerous consequences, as shown by the examples of cases that have occurred in practice in Section 1.1 below. Indeed, among the existing critical sectors, the electrical infrastructure sector is considerably critical because failures in this sector can have negative effects on all other dependent critical sectors [20]. Indeed, a very large number of areas performing critical functions are highly, if not totally, dependent on electrical energy. To name but a few, hospitals or water supply systems are equipped with backup systems but these have limited production capacities and in the catastrophic scenario of a blackout lasting several days, this can therefore also impact those critical infrastructures [21] and cause significant property damage and loss of human life [22].

To cite figures for a concrete example, the authors of [23] studied the effects on mortality of one of the largest power outages to have occurred in the United States (that of 14 and 15 August 2003 in New York [24]). To do this they carried out a statistical study using a generalised linear model using data from 1987 to 2005 and the results are that mortality during the blackout increased by 122% for accidental deaths and by 25% for non-accidental deaths (i.e. disease related). This corresponds to about 90 excess deaths. This increase in accidental deaths and injuries during power outages is thought to be due to carbon monoxide (CO) poisoning, food poisoning and hypothermia, among other things [23].

Different types of cyber-attacks exist and can reach the cyber layer of many critical infrastructures in different fields. In the critical field of energy and more particularly electricity, a number of implementations of these different types of cyber-attacks have already been observed in the world, with sometimes very important consequences on the affected areas. Some examples of cyber-attacks that have taken place against power system infrastructures are presented in the following section in order to see the consequences that poor/non-existent monitoring, control and protection of the cyber-layer can have.
Next, Section 1.2 introduces the CYPRESS project that I had the opportunity to work on during my internship at BlackLight Analytics and that has been the guiding line throughout the writing of this document in which each step has been carried out in order to help the realisation of this project.
Finally, Section 1.3 presents the structure of this Master's thesis and its contribution to the CYPRESS project introduced in the previous section.

1.1 Cyber-vulnerabilities of cyber-physical power systems

In cyber-physical power systems, the occurrence of faults and failures can come from an increasing number of elements. Firstly, there are the so-called traditional failures, directly infecting the power network. Among these are those caused by bad weather such as lightning or falling trees. Or blackouts may be caused by an imbalance between power generation and power consumption. In these cases, if for example the demand exceeds the production, the frequency will decrease and from a certain value the automatic load shedding plan will be activated to avoid power cuts. However, what happens if the frequency decrease is excessive is that the power generation plants will shut down in cascade until the network totally collapses and leads to a blackout [25].

Secondly, there are failures that are more indirect and are the result of failures in the cyber layer of these CPPS. These, as written above, can be unintentional and due to bugs or configuration issues but they can also be of anthropogenic (criminal) origin, with cyber-attacks. In order to understand the importance of research into the cyber security of electricity networks, this section presents some real examples of attacks that have taken place against the ICT network of electricity infrastructures around the world, causing damage of varying degrees of severity. Note that in the context of a cyber-attack, the motivations of attackers are diverse and can range from state-level attacks to financially motivated attacks [26].

According to the authors of [27], the first documented case of a cyber-attack collapsing the power grid took place in December 2015 in Ukraine. This attack was meticulously planned

and executed to perfection, causing a six-hour blackout for about 210 000 people around the Ukrainian capital. During the attack, computers in control centres were hacked and manipulated by "invisible" attackers who, in the eyes of the helpless operators, de-energised about 30 substations, disabled emergency power sources and deleted essential system files, causing computer crashes. In addition to these attacks, it was impossible for Ukrainians affected by the blackout to contact the call centres as they were saturated with false calls.

An example of an attack that is fairly well known in the cyber security community is the Stuxnet worm[1] that was discovered in an Iranian nuclear power plant in 2010 where it caused significant damage to industrial centrifuges used to enrich uranium [28]. This attack has thus greatly slowed down Iran's vital nuclear energy development [29]. This highly complex malware is suspected to have been introduced by plugging in an infected USB stick and resulted in 45,000 computer systems being infected.

Another notorious case is the case of the organised hacker team "Dragonfly", which in 2014 attacked more than 1,000 companies in the European and North American energy sector with the aim of spying on these companies. Dragonfly gained access to power plant control systems through malware in emails, websites and third-party programs. However, in this case, the intrusion was detected before Dragonfly was able to affect the power supply in the affected regions [29].

More recently, in May 2021, a major electricity company in the UK suffered a cyber-attack that prevented its employees from accessing their work email. This was the second attack on an energy sector organisation in two months, as the European network of electricity transmission system operators ENTSO-E was also the victim of a cyber-intrusion in March 2021. Although the distribution of electricity was not impacted in these cases, this raises concerns and questions about whether cyber-attackers could take control and affect supply [30].

The likely severity of these consequences explains why the proper application of cybersecurity practices is very important in order to further expand the smartisation of power systems [19].

1.2 The CYPRESS project

Based on the increasing number of criminal and non-criminal failures in the ICT layer of CPPS and the limited security and safety that these systems have offered against such failures, there is a need for further research. This research should lead to an improved understanding of the interactions between cyber vulnerabilities and electrical stability and a more secure future for the energy sector (and by extension for other sectors, as explained above).

This is why the fundamental research project "Cyber-Physical Risk of the bulk Electric Energy Supply System" (CYPRESS) [31] has been initiated. The objective of this project, which started in November 2020 and will last for 5 years, is to develop new knowledge, methods and tools necessary to guarantee the security of supply through the electricity transmission network. These developments aim to address cyber threats by integrating them into a coherent probabilis-

[1] A worm is a malicious software which replicates itself on multiple computers without human interaction (contrary to the virus) by using a computer network such as the Internet.

tic risk-management approach. The latter (probabilistic risk management) has been explored in the literature and is beginning to be used in practice, but the "cyber-vulnerability/physical grid stability limits" interactions have hardly been explored. This is why this objective of developing knowledge, methods and tools in the cyber-physical reliability management of the transmission system has been initiated [31].

CYPRESS brings together a team of researchers from the University of Liège, the Université Libre de Bruxelles, KU Leuven/EnergyVille and the company BlackLight Analytics. It is as an intern at BlackLight Analytics that my work was linked to CYPRESS, whose first objectives served as a guideline during the writing of my Master thesis.

During this internship, my role has been mainly to carry out research work upstream of the future tasks on which BlackLight Analytics will have a role to play within the project. This research is mainly aligned with the objectives of Task 2.2 "Co-simulation of cyber-physical transmission" of the CYPRESS project. This task consists among other things of selecting suitable platforms for the physical and cyber parts of the power system, integrating them into a co-simulation framework and completing this combination with pre- and post-processors for data input and result output.

Therefore, my tasks were mainly to research existing platforms for modelling and simulating cyber-physical power systems. A small test case was also implemented as a proof of concept of the outcome of the research. In addition, a small part of the research was focused on some possible choices of components to model in the cyber-physical layer of power systems in order to improve the relevance of the simulations as well as the way to make the final platform generic and modular, which is an objective of the project.

1.3 In this Master's thesis

This document is structured as follows: Firstly, Chapter 2 presents a broad state of the art of existing CPPS simulation tools and platforms in the literature. The challenges to combine physical and cyber domain simulations of cyber-physical power systems are first presented in Section 2.1, followed by a presentation of the different solutions that exist to address these challenges in Sections 2.2 and 2.3 with a particular focus on temporal synchronisation in Section 2.4. In the remainder of this chapter, a wide range of tools specialised respectively in power system and communication network simulations are presented in Section 2.5 before concluding on the possible choices for the use of a co-simulation platform.

Secondly, Chapter 3 presents the practical part of the work done for this Master's thesis. Following the tool research carried out in Chapter 2, an implementation of a co-simulation platform has been considered and a use case is first described in Section 3.1. Then, Sections 3.2, 3.3, 3.4 and 3.5 describe the design steps of this practical implementation, punctuated by the results of the test scenarios carried out in Section 3.6.

Finally, Chapter 4 proposes possible developments for future work, especially in terms of genericity, before finally concluding this master's thesis with section Section 4.3. Note that most of the work consisted of research work, which is mainly found in Chapter 2 and somewhat in the conclusion Chapter. The practical part was intended to serve as a proof of concept and as first real implementation tests of the desired co-simulation platform highlighting the feasibility

of the research.

Chapter 2

State-of-the-art of tools for simulating cyber-physical power systems

In order to enable the development, testing and assessment of new protection or optimization techniques and tools, modelling and simulation is a key method that is widespread in the scientific community [5]. Simulation is a powerful tool for evaluating control and protection systems and algorithms [18], with economic and practical benefits. Developing a new technology or upgrading an electrical or communication network (by disrupting the part that needs upgrading) can be quite costly economically and socially, even for a short period of time [32]. The costs of installing and testing power system or ICT infrastructure upgrades as well as the costs of potential loss of service as a result of these upgrades are reduced by simulation [2]. Indeed, if there are errors in what is installed, these costs can become substantial, whereas with simulation we can reduce them by testing technologies and upgrades before deploying the technology. In this way, different configurations can be tested and improved before installing it, allowing for a variety of scenarios and conditions to be analyzed. With cyber-physical power systems, performing a simulation can help to better understand and study the interactions between the cyber and physical domains [2]. Moreover, it is obviously easier to carry out tests and developments in a simulation platform, which corresponds to a well-controlled environment, rather than in a real situation.

There exist a large number of power system simulation tools and a large number of communication network simulators. These tools are commercial or open-source and some of them will be described in Sections 2.5.1 and 2.5.2. For a broader list, the interested reader can refer to [17] and [15]. The problem is that the close interdependence between power and communication systems makes it difficult to analyze them using a single simulator designed to simulate only the power systems or the communication networks. Traditional power grid simulators do not model the communication protocols and traffic patterns found in today's networks [2]. In most of these simulators, the cyber layer is only considered in a very simple way, e.g. by simply assigning constant communication latency when simulating ICT dependent control systems [5]. Concerning simulators dedicated to communication networks, they are not at all designed to model power systems. Therefore, there is a critical need for simulation tools that combine communication and electrical models and enable detailed analysis of the dynamics and mutual impacts of both domains [17],[18],[3].

The creation of such simulation environments for cyber-physical electrical systems has been studied increasingly over the past decade and research and development processes are still un-

derway. This chapter tries to review as well as possible the tools, software environments and methods described so far in the scientific literature. This state-of-the-art should help to make choices regarding the simulation methods and tools to be used and/or developed within the CYPRESS project. These choices are discussed progressively at the end of each section of the state-of-the-art as well as in the conclusion of SECTION 2.6.

In the following, SECTION 2.1 presents the challenges for combining simulations of power system with communication networks. Some solutions to these challenges are given in SECTIONS 2.2, 2.3 and 2.4, containing respectively the global approaches for coupling heterogeneous simulators, the frameworks to interface simulators in a co-simulation and the techniques to enable synchronization between simulators. After that, SECTION 2.5.1 presents tools specialized in power system modelling and simulation while SECTION 2.5.2 presents those specialized in communication networks. The chapter ends with the conclusion in SECTION 2.6.

2.1 Challenges for a combined simulation of both domains

modelling and simulating systems belonging to heterogeneous domains can be a complicated task. The combination of the power system domain with the communication and information domain is no exception to this fact since interfacing and synchronizing these two worlds presents several challenges that are discussed in the following. First, the governing principles for modelling and simulating power systems and communication networks (separately) are discussed. Then, the challenges of combining these two domains in a simulation environment are presented.

2.1.1 Principles for modelling and simulating both domains

Simulators of electrical and communication networks tend to adopt different modelling approaches [2].

In terms of temporal resolution of the electrical domain, dynamic simulations of power system simulators often adopt continuous time modelling [7]. Electrical systems are composed of a set of individual physical components that are assembled in a complex way and represented by sets of differential algebraic equations (DAE) emanating from fundamental physical laws such as Maxwell's equations [5]. Consequently, the system state variables change continuously with respect to time [33] and the relations between these continuous state variables of the components are defined by these differential equations which represent the dynamics of the different elements of the power system [2]. Moreover, the individual components interact dynamically with each other because of their physical coupling, and the state of a component depends also on the states of other components, so that the DAEs are also coupled [5].
On the other hand, there are still elements, such as circuit breakers, that bring discrete dynamics to the system. Moreover, solving these differential equations analytically in order to obtain closed form results is feasible with simple cases, but solving DAEs analytically and exactly is not straightforward for non-trivial real world cases and such closed form results are therefore not obtainable in these cases [33]. Therefore, time steps are used to simulate the electrical domain, using numerical algorithms based on discrete time steps [2]. This leads to the temporal evolution illustrated in FIGURE 2.1.

Note that the methods used by power system simulation tools can be based on steady-state calculations (power flow) or continuous-time calculations (electro-mechanical or electromagnetic simulations) and that it is up to the developer/user to choose which model to use [5]. This aspect is further explained in Section 2.5.1.

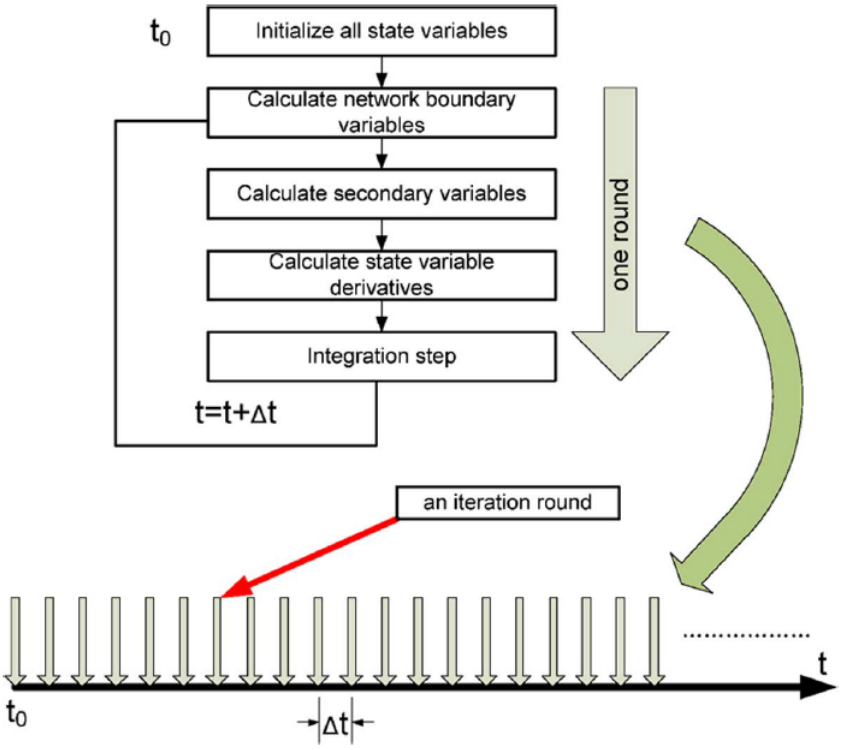


Figure 2.1: Power system dynamic simulation. [1]

On the other hand, concerning communication networks, their modelling is not based on physical principles but rather on the functionality of components. It must be possible to simulate the different communication processes, from the transmission of messages and signals to their processing [5]. This is why, concerning the temporal resolution of communication network tools, these simulators are generally discrete event simulators [7] in which each event is a significant step of the message processing or transmission [5].
These discrete events are for example the sending of packets from a node, their reception at another node, the expiration of timers, the processing of a message on a node, etc. and generally take place in a non-periodic way, which results in the use of an approach where the interval between the events is not fixed contrary to the approach used for power systems [2]. Here, the events are ordered and stored by an event scheduler and processed one after the other during the simulation [2], as can be seen on Figure 2.2.
It should be noted, as was the case for the electrical domain, that the developer has the choice concerning the level of detail of modelling and simulation used. Indeed, globally two types of choices are possible. Either the communication processes are represented by statistical models in which the transmission delays and waiting times during packet processing are determined via random distributions. The second choice lies in the possibility of using a communication system emulation that will model the different functionalities of the system in more detail [5].

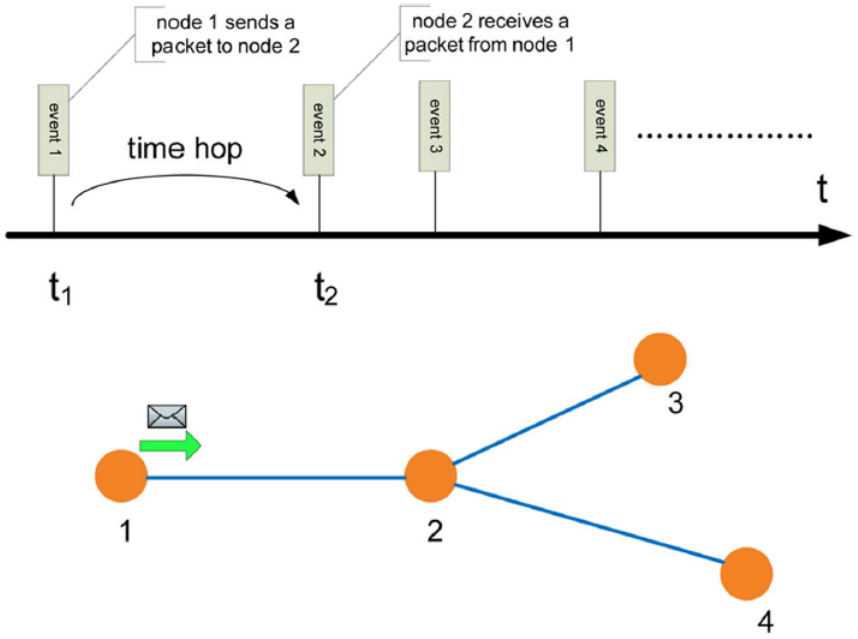


FIGURE 2.2: Communication network simulation. [1]

2.1.2 Challenges

One of the major challenges in the combination of both domains is that they, as explained above, often adopt a difference in temporal resolution, different time models are used by the various simulators, with continuous models on one side and discrete event models on the other. Therefore, especially when combining these tools in so-called co-simulation approaches (see SECTION 2.2.3), the time synchronization of the different components of the co-simulation is a topic of major importance [2] that can lead to difficulties and inaccuracies when simulating the cyber-physical system, as will be discussed in SECTION 2.4 dedicated to temporal synchronization.

This time synchronization challenge is linked to the way to figure out how to detect, link, and process events in both domains. In a power system, the detection of an event can often be due to the crossing of a certain threshold (e.g. a voltage going to zero or a current exceeding a specified value, etc.), which is a very computationally intensive process, especially for large systems [5]. Another event can be simply the end of a power flow calculation and potentially sending some state variable (voltage, current, active, reactive power,...) to be processed [33]. Whether it is with these techniques or with the events generated in the ICT domain, the detection of an event might most probably lead to a transmission of information to the other domain to induce a reaction. It is this information flow that is a crucial point of the coupling because both domains must then execute in a synchronous and deterministic way, avoiding for example that a communication network simulator moves to a future event while the power system simulator is late and has not yet detected or processed the information of the last event sent [5].

Moreover, as written above, the existing tools dedicated to the modelling of electrical networks do not model in detail the communication networks and the same goes for the communication simulators that do not support the modelling of power systems at all [5]. The proposed solutions to achieve the coupling between these two domains are presented in SECTION 2.2.

Other more general challenges in the design of the simulation tool combining the different domains is that it must be able to specify different scenarios in a flexible way and one must also be able to define the level of detail of a simulation (by playing for example on the temporal resolution). Moreover, scalability is also an important feature, the simulator must be able to adapt to complex scenarios applied to large-scale networks.

We can note that, as written by the authors of [5], when building a software whose goal is to combine the power system layer and the ICT layer, a trade-off exists between three options out of which we can generally choose only 2. These options are reusability, precision and speed of execution. If one wants to simulate smart grids accurately and quickly, one cannot use 100% reuse as it is, since one has to build the simulation environment of the hybrid system and, if one does reuse existing tools, one has to modify them. On the other hand, if we want a high level of reusability, the simulation will be based on ad-hoc approaches to allow synchronization and interactions, which results in calculations that can either be fast or accurate, not both. The choices will therefore depend on the context and the desired application.

2.2 Solutions for a combined simulation of both domains

First, it is important to note that to date and according to the research done so far, no tool or environment for the combined simulation of cyber-physical power systems has been found that was created from scratch and does not use at least one existing tool. Indeed, it is very complicated, time consuming and expensive to build from scratch an entire tool combining ICT components and power systems [2]. Therefore, the approaches developed so far use existing simulators and join them in some way to achieve the desired simulation [2],[18]. These (open source or commercial) simulators have the advantage of being mature and efficient as they are developed over multiple decades and widely used in the domains for which they have been created [33].

Looking at the literature, it turns out that there are different types of approaches to couple the heterogeneous simulators. The first step is to choose one of these approach based on the characteristics, advantages and drawbacks they offer, which are given in this section.

2.2.1 General purpose tools

With tools such as MATLAB, it is possible to simulate both types of models in one environment [16]. However, as these tools are not specifically dedicated to the simulation of power systems or communication networks, they lack efficiency, do not provide enough models or modelling libraries and are therefore not optimal compared to specialized tools [5].

2.2.2 Integrated or comprehensive approach

Another approach enabling to simulate both types of models in one environment is the integrated or comprehensive approach [18]. With this approach, both domains, *i.e.* power system and communication network, are modelled and simulated in a single environment as can be seen in FIGURE 2.3 [18],[2]. The main challenge is to combine the two types of models in one environment, e.g. by implementing the power system techniques in an ICT environment or vice

versa [18],[2]. This kind of tool must provide a single simulation interface instead of one for each simulator, which is beneficial for users. This feature is a challenge as the single interface must offer the ability to build detailed models of smart grid simulations [2].
It seems that integrating a simulator for one domain into another simulator for another domain must be very complicated in practice. You have to take into account all the models and all the simulation techniques of the integrated domain. It will surely never be easy to have an integration as powerful and complete as a simulator dedicated to the domain as this method leads to many simplifications and a potentially inexact and deficient translation for foreign models [7].
An advantage of integrated simulation environments is that no time synchronization is needed as both models use the same logical simulation time in the same time domain [18]. A disadvantage is that they rarely offer the possibility to simulate complex scenarios: they are limited in the complexity of simulation scenarios and lack efficiency in these cases [8]. According to [18], this kind of approach is well suited to analyse stationary power system (e.g. power flow) but is less adapted to dynamic power system simulations because those are much more complex.

This approach can be found in [34], where they use OMNeT++ (see Section 2.5.2) as development platform to build an integrated smart grid simulation framework. They use existing models for communication networks and add power grid models: they have modelled an electrical distribution system in MATLAB and linked it into the OMNeT++ platform [34].
In [35], the authors build a *Smart-Grid Common Open Research Emulator* (SCORE), which is a smart grid emulator built upon an existing open source communication network emulator, CORE [36] and that uses a power module emulating DC power flows as power system emulator. SCORE mainly focuses on real time simulation [35] and allows to port solutions developed in the environment to embedded devices with few modifications or no modification at all [37].
Other integrated platforms are given in [38], where the Network Simulator NS-2 (cf. Section 2.5.2) is extended to simulate physical system dynamics and controls or in [39], where OPNET Modeler is extended to simulate wide area communication networks (WAN) in power systems.

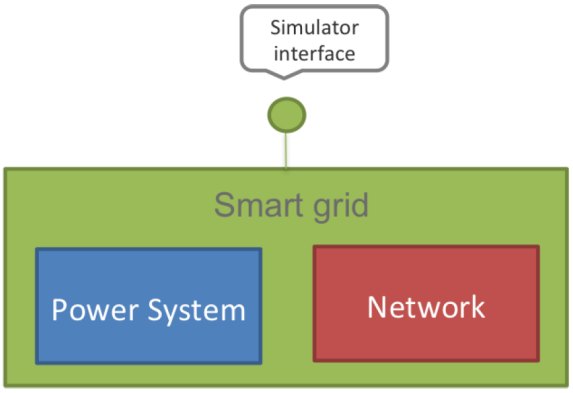


Figure 2.3: Integrated simulation approach. [2]

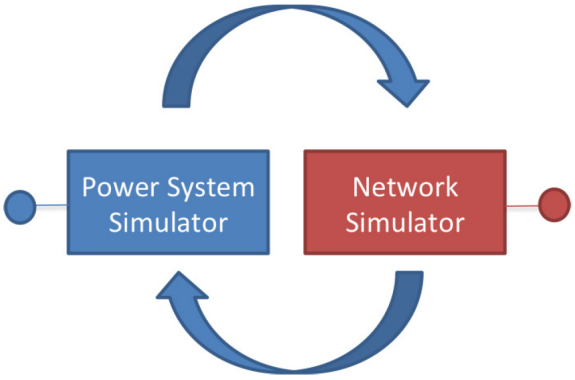


Figure 2.4: Co-Simulation approach. [2]

2.2.3 Digital co-simulation approach

In the case of digital co-simulation, existing simulators specialized in both domains are joined in order to realize a purely virtual cyber-physical co-simulation, as can be seen in Figure 2.4 [2]. A big challenge with this approach lies in the need to synchronize the tools accurately

and to enable interactions between them [8]. According to the authors of [40], this fully digital approach lacks the ability to handle unknown and cascading power system and ICT network events. They use the example of EPOCHS [3], a disadvantage of which is the potential accumulation of errors due to the way it implements synchronization (cf. Section 2.4) and GECO [33],[1], which can also induce simulation errors although removing some that used to be induced in EPOCHS.
However those two particular tools were last updated in 2006 and 2012 respectively and several other co-simulation tools with other characteristics have been developed and updated since then.

Note that most of the time, for this kind of approach to work, the combined tools must present adequate Application Programming Interfaces (APIs) [5]. It may happen that some simulators do not offer such a feature. In that case it is really hard or even not possible to use them in a co-simulation while in some other cases, the available API just needs some modifications to connect, synchronise and manage the objects of a certain tool [18], making the co-simulation feasible.
According to [41], identifying the most suitable heterogeneous tools to be coupled requires modelling experts from the corresponding domains in order to select those with the capabilities required for a certain simulation objective. In addition, compared to simulations involving hardware (discussed below), digital co-simulation would be limited in modelling complex and various electrical equipment, communication equipment and private communication protocols according to authors of [40]. Another drawback pointed out in [2] is that performance penalties and delays may infer from the necessary synchronization.
Regarding some advantages of co-simulation approach, it reduces development time and modelling effort as existing tools are used and there are "just" additional features to be added, which reduces the implementation error risk [2]. Furthermore, a co-simulation can run in a distributed manner across different runtime environments and there is the possibility to build generic systems that can be reused more easily with less effort needed to interface the different simulation components in other applications [41].

The first known approach combining simulators for power systems and communication networks [18] is EPOCHS [3]. This tool uses the co-simulation approach to combine NS-2, the PSCAD/EMTDC electromagnetic transient simulator (cf. Section 2.5.1) and the PSLF electromechanical transient simulator (cf. Section 2.5.1). This combination allows the user to choose whether to perform the analysis of electromagnetic scenarios with communication (using NS2 and PSCAD/EMTDC) or electromechanical scenarios with communication (using NS2 and PSLF). EPOCHS uses a specific approach with a core component that enables the coordination and synchronization of the different simulators, this aspect is discussed more deeply in Section 2.3.1.

Another more recent project (last updated in April 2021), called "Smartgrid Cosimulation Platform", is based on co-simulation and couples together the NS-3 (cf. Section 2.5.2) for network activity and OpenDSS to simulate power flow calculations (cf. Section 2.5.1). The combination between these simulators is done via a co-simulation framework called mosaik [42] which is presented in Section 2.3.2 and an example of transformer on-load tap control is put into practice with this platform [43].
Note that many other works have been done with the co-simulation approach and some of them are described in the following sections.

2.2.4 Hardware-in-the-loop approach

Some approaches use real equipment to replace part of the simulation models [40] and tend to reduce some of the disadvantages of digital co-simulation. These are called Hardware-in-the-loop (HIL) simulations. A HIL simulation characterizes a digital simulation of the operation of an embedded system for which the hardware environment is simulated: the inputs and outputs of the embedded system under test (hardware) are connected to a computer that reproduces the operation of the environment (plant simulation) [44]. Electrical emulation of sensors and actuators are included in the simulation thanks to a mathematical model of the simulated system [2] and make up the interface between the plant simulation and the hardware under test [45].
A HIL co-simulation allows testing of hardware and software components under more realistic, detailed and accurate conditions [5],[40]. It gives more accurate and closer to reality results [46] as potentially erroneous or incomplete virtual models are replaced by real components [5]. However, a challenge raised in [5] is that this type of simulation must guarantee the execution of the virtual simulator in accordance with the real-time constraints of the tested hardware.

A HIL simulation platform of cyber-physical power system is introduced in [40] where they use this platform to simulate a system that is subject to a cyber-attack. The co-simulation system is composed of DIgSILENT (cf. Section 2.5.1) for power grid simulation and a combination of communication simulation software QualNet (cf. Section 2.5.2) and three real routers and they implement an automatic voltage control (AVC) algorithm as power controller. In [47], they develop a real time HIL test platform whose goal is mainly to emulate the behaviour of a realistic Active Distribution Network, *i.e.* an electrical grid of which the energy resources are actively controlled by an energy management system. Their platform is built with the eMEGAsim PowerGrid Real-Time Digital Simulator provided by Opal-RT.

2.2.5 Choice

A choice among these different approaches must now be made. First of all, general purpose tools (such as MATLAB) do not bring enough detail and efficiency compared to specialized power or ICT tools, so that they will not be chosen here. Secondly, regarding the possibility of using hardware to perform the simulations, this is not really feasible in the context of my Master's thesis, where I will be limited to purely virtual simulations. Indeed, I do not have access to any hardware, which would obviously be necessary to carry out hardware-in-the-loop implementations. Regarding the choice that the CYPRESS collaborators might make, adding hardware to the simulation loop is clearly a possibility as they plan to create a test lab in the project. However, it seems that a purely numerical simulation process is also planned, so the choice will be up to them, depending on what is planned. By the way, the goal here is not to try to perform real-time simulations as some tools offer, so this is not a criterion that will help to choose.

The final choice (for my Master's thesis) stands between creating a single simulation environment integrating both types of systems while providing a single interface or creating a co-simulation by joining separately specialized simulators that will then have to be synchronized and for which it will be necessary to establish the possibility to interact between them. A preference choice in the majority of the works found during the literature research is towards the co-simulation approach and it is also towards this one that I am going to move. As it

has just been said, there is more work done on this method, which offers a greater number of possibilities to analyze and potentially use. In addition, co-simulation is more practical and should offer greater possibilities for making generic and modular platforms [48]. Finally, the limitation regarding the complexity of the possible simulation scenarios with an integrated approach, stated in two papers, does not seem to make it an interesting enough candidate for the CYPRESS project where the power grid and the accompanying communication layer will have to be modelled in detail and simulated effectively.

Since the main challenges with co-simulation are to enable interaction and data exchange between simulators and their temporal synchronisation, the remainder of this chapter is organized as follows: Different ways of making the simulators of a co-simulation interact with each other are described in Section 2.3 in order to know which type of architecture to use to join the simulations in an efficient way. Existing methods to synchronize the tools in the chosen framework are discussed in Section 2.4. Section 2.5 proposes an overview of some simulation tools specialized in power system or in communication networks so that the more adequate ones can be chosen. Finally Section 2.6 concludes the chapter by summarizing the choices that have been made through the previous sections.

2.3 Frameworks for coupling co-simulation components

The authors of [5] list four different frameworks and techniques introduced in the literature to interface simulators of both domains while those of [4] classify the existing methods for simulators coupling in two parts, themselves divided in subparts. Based on both papers as well as on other ones (that do not necessarily carry out a methods' overview), this section gives an overview of several frameworks to couple and interface simulators in a co-simulation.

2.3.1 Co-simulation standards

Before presenting these different approaches, it is important to note that there are standards in the field of combined simulation. These standards have been implemented and used in some works, are potentially coupled to the approaches presented below and are generally chosen for the features of reusability and genericity that they bring.
The most common standards used in the literature regarding co-simulations of cyber-physical power systems are the *High-Level Architecture*, which provides an interface and a set of guidelines to direct the execution of a co-simulation, and the *Functional Mock-up Interface*, which provides an interface for communicating with the simulation models [49]. Both are described here-below.

IEEE 1516 High Level Architecture (HLA)

The high-level architecture is a software architecture specification (a standard) that provides a general framework within which simulation developers can structure and describe their simulation applications. Its purposes are to allow flexibility, interoperability between simulations, and reuse of models in different contexts [50]. This architecture defines how to create a global simulation composed of distributed simulations that interact without being recoded [51].

In an HLA-based simulation, each participating simulator is called a *federate* and it interacts with other federates in an HLA *federation*, *i.e.* a group of federates [51]. The federation is done

thanks to a core component called the *Run Time Infrastructure (RTI)*, serving as a central administration instance [18]. The RTI provides a set of software services that are needed to help federates coordinate their operations and data exchange during an execution, it enables synchronization of simulations and routing of communications between federates [3]. Furthermore, a *Federation Object Model (FOM)* is defined for the overall simulation [18]. It is a specification that defines the information exchanged at runtime such as Object Classes and Interaction Classes [50],[52]. HLA also provides a generic object-oriented model description for realising a synchronized and distributed environment. This standardized model description is called *Object Model Template (OMT)* [18]. Based on the latter and in order to enable bidirectional communication from and to the RTI, the RTI provides an API. This API must be implemented by the federates and will make it possible to access "system-wide and simulator-specific attributes and interactions" [5].

An advantage of this architecture is its scalability: one can perform highly paralleled simulations of large-scale systems using HLA frameworks [5]. Moreover, the concept of federates means that several types of components can be combined, for example simulation models but also concrete functional codes and potentially hardware. The main advantages provided by HLA are interoperability and reusability [53].
On the other hand, among the challenges and disadvantages it presents, the implementation of standardized interfaces to simulators may require significant work. Indeed, to make existing simulators compliant with HLA specifications, modifications must be made and this can be difficult in some cases. It is particularly complicated in the case where the source code is not available (for example with commercial off-the-shelf simulators). In this case it is possible to make the simulators compatible with the HLA federation but it is often quite tricky, notably due to the large number of additions to be made, the limited number of input and output options offered by these tools and the difficulties in accessing their internal simulation state [3]. Other drawbacks with HLA-based frameworks lie in the cost of licenses to use advanced RTI, even though open source implementations of HLA are available [49],[5]. In addition, since the system operates using a publish-subscribe mechanism to exchange information [52],[53], any information that is updated by a federate will be received by federates that have subscribed to it. When this is not necessary, it will cause unnecessary delays and exchanges [3].

A first example of a tool based on HLA is EPOCHS [3], which was already introduced in Section 2.2.3. EPOCHS connects power (PSLF or PSCAD/EMTDC) and communication (NS-2) simulators using a RTI based on the HLA (IEEE 1516-2000) [54], so that it works in the same spirit as HLA but with a custom interface for a simpler implementation [3]. The RTI of EPOCHS allows the routing of communications between the components. The last component, as can be seen in Figure 2.5, is AgentHQ, providing a unified environment for the agents[1].
Basically, none of the used simulators are designed for interoperability, so they had to find a way to federate them. In [55], the different methods available to federate commercial off-the-shelf components are presented. In EPOCHS, internal APIs have been used to federate all the simulators, all of which allow user-defined extensions [3].
They added a new transport protocol to NS-2 in order to establish the link to the RTI, and this protocol is invoked once per time step to stop execution and interact with the RTI.

[1] In [3], they define agents as "autonomous, interactive computer programs capable of communicating over a network". The agents of interest may be located in a number of intelligent electronic devices (IEDs) that enable control and protection. These agents can set/receive values from the power system as well as receive and send messages to each other.

PSCAD/EMTDC generates FORTRAN source code from scenarios generated in its graphic environment. The tool can thus be extended by making calls to the source code in C or FORTRAN. Concerning PSLF, with the help of the EPCL language[2], a stub is created, allowing the interaction with the RTI by using files [3].
The advantage of EPOCHS using this type of approach is that it uses non-intrusive techniques to federate simulation engines using only their built-in APIs [3]. Note that according to [29], EPOCHS does not support cyber-attack simulation.

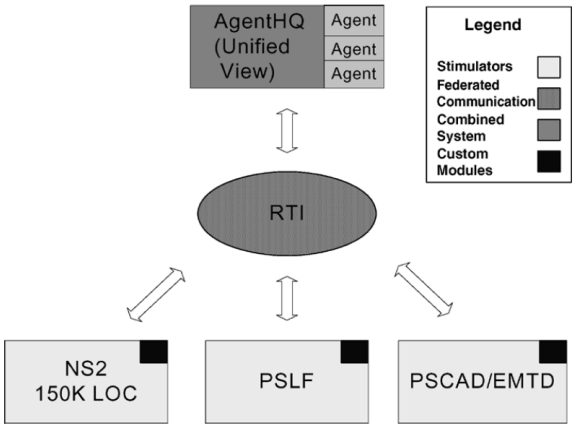


FIGURE 2.5: Relationship between EPOCHS' components. [3]

Note that IEEE 1516-2000 - *HLA* was created in 2000 and that an evolution was created in 2010, *i.e.* IEEE 1516-2010 - *HLA evolved* [52]. Furthermore, note that there exists an open source HLA/RTI implementation called OpenRTI [56].

Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) is "a free standard that defines a container and an interface to exchange dynamic models using a combination of XML files, binaries and C code zipped into a single file" [57]. It facilitates the exchange of models across modelling tools [58] and defines abstract functions that each component of the simulation must implement, which ultimately provides a common interface. The FMI standard also defines Functional Mock-up Units (FMUs), which are black box packages of models. These FMI components can be plugged into a larger model, *i.e.* a combined simulation, thanks to the standard interface [5]. In addition to the models, solvers can be included in FMUs [59]. Then a FMU has two modes: either it provides access to the model equations (Model Exchange), or it implements a solver that manipulates the model equations (Co-simulation). The first mode is more similar to the approaches integrating the different models into a single tool such as those presented in SECTION 2.2.2 [4].
An advantage, that is also a HLA feature, is the possibility to perform simulations in a parallel and distributed way, which can result in faster simulations than for single-threaded cases [5]. Moreover, the standard is broadly supported by modelling tools [49]. Note that simulators interfaced via FMI also need a co-simulation master (cf. SECTION 2.3.2) that enables their coordination and synchronisation [41].

The authors of [59] have developed the DACCOSIM co-simulation environment [58] based on the FMI standard. This tool, which evolved into DACCOSIM NG in 2018 [60], allows to

[2]EPCL is the PSLF's proprietary language.

build and run co-simulations and provides a graphical interface to design co-simulation graphs. FMI compliant simulators can be integrated into DACCOSIM NG. The co-simulation platform CyDER introduced in [61] is also based on the FMI standard and includes different domain-specific simulators in order to allow planning and operation of the power grid. However, this tool does not integrate ICT simulators.

Some implementations using the HLA to realize a cyber-physical system simulation also use the FMI simulation interoperability standard. Paper [62] proposes SAHISim, the *Standardized Architecture for Hybrid Interoperability of Simulations*, a framework for distributed simulation. SAHISim is based on HLA and FMI: the federates used can be FMUs and an advantage of this platform is that it offers the possibility to run the different components in parallel. The use of HLA and FMI allows for a standardised format [62].

2.3.2 Coupling with Orchestrator

Concerning the co-simulation approaches properly speaking, the first one presented is the co-simulation with "orchestrator". In this one, a master algorithm is set up to orchestrate the framework by instantiating, interfacing and coordinating the simulators as can be seen in the example of FIGURE 2.6. This central component has to configure and initialize the simulators, synchronize their time during the whole simulation and exchange variables and events between them [7]. The communication between these simulators is done either via a message bus governed by the master or directly from one to the other following a route defined by the master algorithm [4]. According to [4], this type of co-simulation simplifies the architecture of the co-simulation framework.
To develop this master algorithm, it can be implemented with a script language such as Python or Java, but also with software dedicated to co-simulation such as the those presented below. These middleware are used to manage the interoperability within a co-simulation and present the advantage of being flexible as simulators can be easily added or removed without affecting the rest of the co-simulation [63]. According to [63], this flexibility is possible thanks to the fact that simulation software have to be interoperable with the simulation's master rather than directly interoperable with each other.

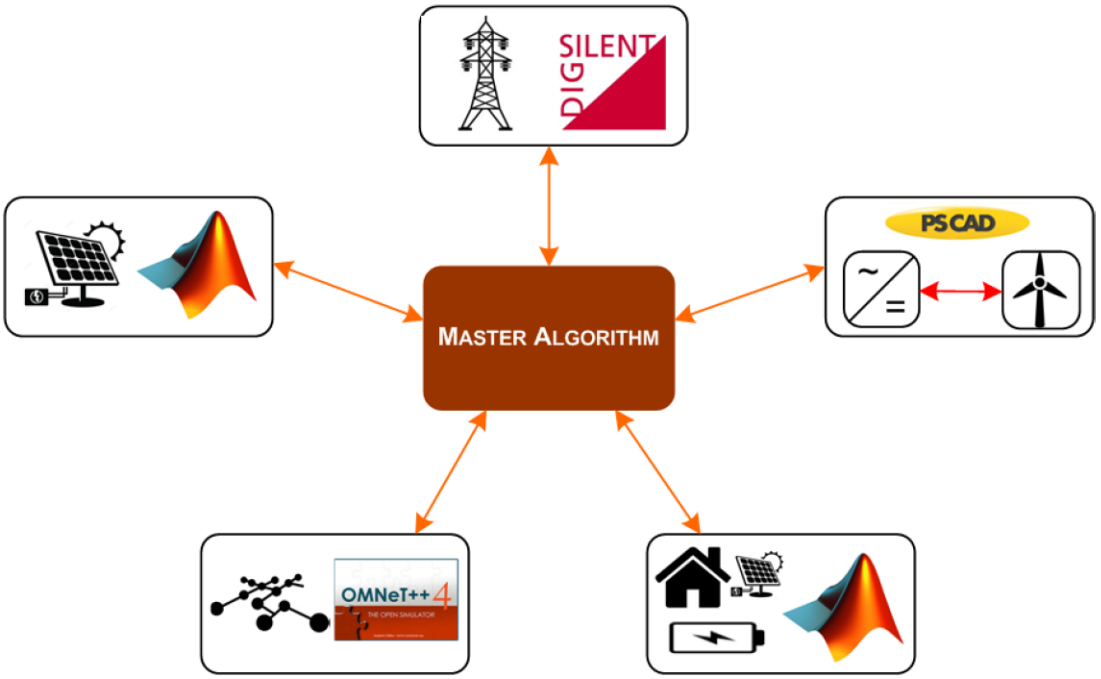


FIGURE 2.6: Co-simulation with an orchestrator. [4]

Mosaik

Mosaik is a flexible, generic and modular open-source co-simulation framework that focuses on energy systems [17] and that is based on an automatic composition of existing simulation models [64],[2]. It is a Python-based framework that enables to couple heterogeneous simulators and control applications and to generate large-scale scenarios [5]. Indeed, Mosaik provides support for specifying scenarios, thanks to a scenario-API, and analyzing the results of these scenarios [2]. These scenarios are generated using the MoSL language (Mosaik Specification Language), thanks to the information provided by the different components of the simulation. The coordination of execution of the different simulators is done thanks to a discrete event-based approach. With this approach (based on SimPy[3] [65]), each simulator may have a different step size and it is possible to control these step sizes at runtime [5],[42]. Mosaik's core contains a scheduler module thanks to which the data exchange between simulators is coordinated at runtime [41] and that allows the platform to carefully track dependencies between simulators by letting them run a simulation step only when necessary (e.g. if data is requested by another simulator) [9].

An advantage of Mosaik is that it has a "component-API" that allows the integration of existing simulation tools regardless of the language in which they have been implemented [42]. In addition to the wide variety of simulators that can be integrated with Mosaik thanks to this component-API specification, some simulation tools that are part of the Mosaik ecosystem are already interfaced with the framework (e.g. PyPower, PandaPower, PowerFactory or Opal-RT eMEGAsim). Furthermore, an FMI adapter is also included in the environment, which allows it to be used as a master algorithm for simulators compliant with the FMI standard (cf. SECTION 2.3.1) [41]. According to [2], the simulation interface of Mosaik should allow a communication network simulator to be integrated in the environment. Indeed, this is possible and several tools, described below, have achieved it.

[3] *SimPy is a process-based discrete-event simulation framework based on standard Python* [65].

The authors of [66] have created, as part of the ERIGrid project [67], a combination of Mosaik with NS-3 (see SECTION 2.5.2) and PowerFactory (see SECTION 2.5.1) as well as MATLAB/Simulink to implement smart grid control applications. They use the FMI standard: the simulators are interfaced via FMI in order to remain generic and the "component-API" must be mapped to FMI so that Mosaik can access the simulators encapsulated in the FMUs (to do this they use a certain "FMI++" toolbox).

In the project "LarGo!" [68],[69], they combined PowerFactory and OMNEST (commercial version of OMNeT++ which is presented in SECTION 2.5.2) around Mosaik. The goals of this project are to prepare for the mass deployment of software applications for smart grids, such as energy management at the customer level or monitoring and control for DSOs and to analyze the technical side effects of deployment, update, patching and operations on a common communication infrastructure [70].

Mosaik was also used as a co-simulation middleware in Astoria[4] [37]. This tool results from the integration of PyPower as a power flow simulator to model the power grid and NS-3 to simulate the behavior of SCADA[5] components and their communication infrastructure. Astoria allows the specification of smart grid networks, as well as the simulation of attacks and the evaluation of their impact on these infrastructures: one can specify precise topologies of smart grids, create different types of attack profiles (a set of attack profiles are provided and it is possible to develop new ones) and inject attack instances during execution.

Ptolemy II

Ptolemy II [73] is an open-source Java-based framework for simulations of real-time heterogeneous concurrent systems [74]. It gives the possibility to combine, among others, discrete and continuous models [17] by offering different models of computation. Thanks to this, Ptolemy II seems suitable for modelling cyber-physical systems since physical properties may be described with continuous time computations while discrete events computations may handle software and control [53]. In Ptolemy II, components are encapsulated as actors that communicate with other actors through ports. A director orchestrates the exchange of data between actors and advances time for individual actors. Actors contain the action performed on input data to produce output data and this data is encapsulated in what they call tokens. According to the authors of [62], Ptolemy II supports the FMI standard. According to those of [15], Ptolemy is part of the cyber-physical co-simulation tools useful for evaluating the cyber-physical security of CPPS, which simulate power and communication systems together.

An interesting feature provided by Ptolemy II that is pointed by the authors of [74] is that it offers the possibility to study the effect of lost packets or of the latency in a communication network on the performances of a control system (in this case, the considered system is a building control system).
In the same paper, they use Ptolemy II to create a software environment that allows various simulators to be connected to exchange data during execution. In this environment, an actor

[4]Note that unfortunately, according to the GitHub page of the software [71], there seems to be a problem that has still not been solved and its use as is seems compromised.

[5]SCADA stands for *Supervisory Control and Data Acquisition*, this is a large-scale remote management system that can process a large number of remote measurements in real time and remotely control technical installations [72].

can be a Java class communicating at runtime with a simulation program. The actor's input token can, for example, be a command signal that needs sending to an actuator in the simulator while a sensor value received from the simulator can be an output token. For the co-simulation they joined a Java package to the framework which contains the actors *Simulator* and *System-Command*. The *Simulator* class is the actor dealing with the start of a simulation program and which deals with the sending of its input tokens to this simulation program and with the reception of new values from it that must be sent to the output port.

The authors of [75] created VirGIL, a modular co-simulation platform aiming to analyze the interactions between demand response techniques, building comfort, communication networks and electrical system operations. To do so, they combined OMNeT++ and PowerFactory using Ptolemy II. In VirGIL, the link between the orchestrator and the simulations of power, buildings, communication networks and control and optimization is done via FMI. FMUs are used for PowerFactory and OMNeT++, where the FMU acts as an interface to the OMNeT++ API and Ptolemy synchronizes the data of all FMUs.

FNCS

Framework for Network Co-Simulation (FNCS) [76],[77] is a framework that uses a federated approach to integrate power grid and communication network simulators. This framework makes it possible to co-simulate transmission and distribution power grid simulators with a communication network simulator. Each simulator is executed on its own process and FNCS takes care of the synchronization and delivery of messages between simulators [78]. Note that FNCS, while being built in C++, offers interfaces for other languages such as C, Java, Python and MATLAB [79].

The synchronization technique developed, consisting of speculative decisions on when the simulators will exchange messages, allows messages to be delivered quickly, consistently and on time between simulators. In particular, their approach allows the communication network simulator to immediately inject messages into its network without adding delays by the framework [78]. In the FNCS presentation paper [76], they integrate GridLAB-D (power distribution), PowerFlow (power transmission) and NS-3 (telecommunication) using the framework. On the other hand, MATPOWER is used for transmission part in their introductory video [78].

GridAttackSim [80] is a smart grid attack co-simulation framework combining GridLAB-D, NS-3 and FNCS [29]. This tool allows to simulate the characteristics of the smart grid infrastructure with several cyber-attacks and to automatically visualize their consequences. GridAttackSim contains a set of predefined attack profiles but it also allows users to design new ones: the framework can be extended with additional libraries of attack models and additional simulation scenarios. The role of FNCS is to allow communication between GridLAB-D and NS-3 and the synchronization of the simulation with configurable time steps. Note that in [81], the creators of GridAttackSim have researched co-simulation tools dedicated to smart grids, and according to their conclusion, the proposed combination is a "promising direction" for studying smart grids. Furthermore note that GridAttackSim is one of the most recently developed tool that could be found (last updated in March 2021).

According to the developers of HELICS (cf. Section 2.3.2), FNCS misses key capabilities such as co-iteration within a time step in order to ensure convergence among federates. In addition, they raise the fact that the software has scalability concerns. They tried to federate

tens of thousands of federates but the performance was considered too poor, so they preferred not to use FNCS as the basis for their own software [79].

MECSYCO

MECSYCO [82] is a co-simulation middleware allowing to interconnect various pre-existing and heterogeneous modelling and simulation (MS) tools. This software executes co-simulation in a parallel, decentralized and distributed way [63] and its co-simulation engine is based on DEVS[6] so as to integrate tools using different formalisms. In consequence, heterogeneous tools can be co-simulated in a homogeneous way under the form of a DEVS system. The middleware is compatible with the FMI standard, the authors have built DEVS wrappers for FMU components.

In [84], the authors simulate a connected home scenario in which a smart meter is linked to a heat pump that can be connected/disconnected via commands (communication). They use NS-3 for the telecommunication network, an FMU for the distribution network and the heat pump and the decision system is a PLC written in Java.

Note that MECSYCO is also one of the most recently developed tool that could be found (last updated in January 2021).

HELICS

The Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS) [85] is an open-source and modular high-performance co-simulation framework [86] that enables high scalability [79]. It allows several existing simulation tools (and/or several instances of the same tool) to exchange data during execution and ensures their temporal synchronization so that they form one single large simulation [86]. The basic idea of HELICS is to integrate transmission, distribution and communication tools but it also allows to simulate the market. This framework supports the FMI standard to provide direct access to a set of existing models and controls and adopts features of the High-Level-Architecture, notably in terms of the time synchronization approach [79].

The main uses of HELICS correspond to the energy field. Indeed, the elements constituting this domain benefit from a large and growing support, with the availability of a wide range of electrical, communications and control systems, transportation or even buildings. HELICS enables the study of scenarios and systems ranging from dynamic transient dynamics to long-term planning studies and steady-state power flows. It also allows to study the markets and the behavior of individual devices or to perform simulations on a national scale with distribution and "bulk" systems [86]. Note that HELICS is also one of the most recently developed tool that could be found (last updated in April 2021).

In [87], they develop a cyber-physical co-simulation code using HELICS to which NS-3 and GridLAB-D are coupled, among others. Their goal is to study the impact of cyber security attacks on distributed energy resource management systems.

[6]The Discrete Event System Specification is a formalism allowing to model and analyze discrete event systems, continuous state systems, and hybrid continuous state and discrete event systems [83]. It converts continuous dynamics (e.g. power systems) into discrete events using mechanisms that detect state events, such as a voltage crossing zero, for instance [33].

Co-simulation orchestrator	Language	Last update
Ptolemy II	Java	June 2018
FNCS	C++	December 2018
MECSYCO	Java and C++	Under active development
HELICS	C++	Under active development
Mosaik	Python	Under active development

TABLE 2.1: Comparison of co-simulation orchestrators depending on the writing language and their last update.

2.3.3 Ad-hoc Approaches

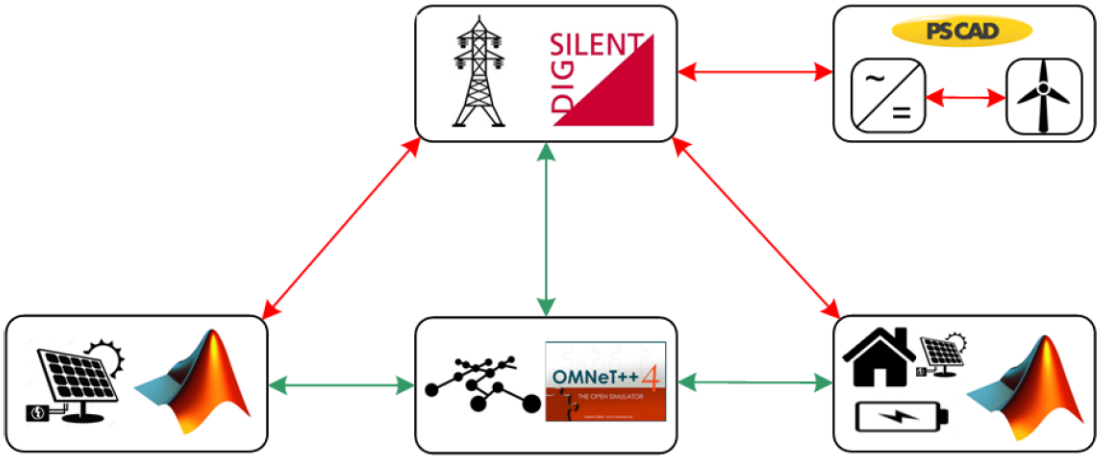


FIGURE 2.7: Ad-hoc co-simulation. [4]

Another way to achieve interoperability is the ad-hoc way, *i.e.* generally by directly modifying simulation tools to make them compatible [63]. This approach is represented by the authors of [4] in the example of FIGURE 2.7. Rather than using a co-simulation orchestrator, interfaces between simulators can be implemented within themselves. This method has the advantage of being concise and efficient. As can be seen from the number of researchers who have implemented it in the table of FIGURE 2.8[7], it is an approach that is often used in research work[8] [5]. In an ad-hoc approach, there is no orchestrator that manages the interactions and interdependence between simulators, which must be configured with great care in order not to lead to a deadlock or inaccurate results [4]. This type of integration allows a specific study to be carried out fairly quickly according to [4].

[7]Note that this table comes from [5], so that the references appearing in it do not correspond to those of the current Master's thesis.

[8]Note that this list does not take into account the projects realised using existing co-simulation masters such as Mosaik or Ptolemy II.

A disadvantage with this type of approach is that it is potentially necessary to make deep adaptations to the tools used. It is necessary to find out how to integrate the different domains (continuous or discrete-event), how to allow synchronisation between the different platforms as well as how to obtain acceptable scalability and usability while these issues are easier to solve with HLA and FMI for instance [5]. According to [4], ad-hoc approaches are rarely optimal for large scale systems. Interface configurations are simulator specific and if one wishes to reuse a simulator whose interface and model have been implemented as part of an ad-hoc coupling, this requires significant adaptations and is therefore not easy to achieve. The solution is potentially to use the FMI standard presented in SECTION 2.3.1. To summarize, this method is most of the time case-specific and is very limited in term of reusability [4]. Several co-simulation works using an ad-hoc framework are given in this section.

	Recent Focus	Power System Simulator	Network Simulator	Simulation Frameworks	Time Strategy	Scalability	Execution Mode
GECO [17][18]	PMU-based WAMPAC, high voltage grid	PSLF	NS-2	Ad-hoc (TCL linking)	Global event-driven	Large systems	NA
INSPIRE [19][20]	WAMPAC, high voltage grid	DIgSILENT PowerFactory	OPNET Modeler	IEEE 1516-2010 (HLA evolved)	Dynamic time stepped	Large systems	NA
EPOCHS [21]	Multi-agent protection and control systems	PSCAD/ EMTDC, PSLF	NS-2	IEEE 1516-2000 (HLA)	Fixed time stepped	Large systems	NA
ADEVS [22]	WAMPAC	ADEVS	NS-2	Ad-hoc (integrated in NS-2)	DEVS	Large systems	NA
VPNET [23]	WAMPAC	VTB	OPNET Modeler	Ad-hoc (Sockets)	Time stepped	Small systems	NA
GridSim [24]	WAMPAC	Powertech TSAT	GridStat	Ad-hoc	Fixed time stepped	Components can be distributed	Real-time + HIL
PowerNet [25]	Controlling power devices	Modelica	NS-2	Ad-hoc (Unix named pipes)	Time stepped	Small systems	NA
Bergmann et al. [26]	Evaluation of DERs and VPPs	NETOMAC	NS-2	Ad-hoc (JNI)	Time stepped	Small systems	Close to real-time
Babazadaeh et. al. [27][28]	WAMC, HVDC, low voltage/mid voltage Grid	OPAL-RT	OPNET SITL	Ad-hoc, emulated, sockets	Real-time	Small (HIL), medium (emulated)	Real-time
Greenbench [29]	Cyber security, low voltage grid	PSCAD	OMNeT ++	Ad-hoc (IPC)	Global event-driven	Tested for small systems	NA

FIGURE 2.8: Overview of State-of-the-Art Co-Simulation Approaches [5]

ADEVS

ADEVS stands for *A Discrete EVent System simulator*. This is a C++ library that is used to construct discrete event simulations [88]. The ADEVS approach does not take the form of a federation and it requires its own modelling of the continuous dynamics of an electrical system [33]. Indeed, mathematically, ADEVS is based on the modular and hierarchical formalism Discrete Event System Specification (DEVS) [88].

ADEVS enables the simulation of control and communication in wide area control scenarios thanks to THYME (Toolkit for Hybrid modelling of Electrical power systems), an electrical system model that has been built for the ADEVS framework [5],[89]. THYME is also a C++ library and both libraries are to be used together [89]. This toolkit offers a number of models of electrical system components, a power flow model and a (limited) model for electromechanical transients [2]. These models can be combined with discrete event process models such as control algorithms or communication networks [5]. By the way, in order to realize models including

a communication network, the NS-2 and OMNeT++ communication network simulators have been integrated to the ADEVS-THYME pair in two implementations presented respectively in [90] and [91]. In [90], they perform a use case of wide area load control and deduce the effects of the communication network performance on the power grid operation, such as the fact that latency impacts the control.

Since in this co-simulation method, the continuous time dynamics is encapsulated within a discrete event model, it provides the advantage of enabling accurate treatment of interactions between discrete event and continuous subsystems. Furthermore, the fact that the synchronization is based on discrete event system modelling makes it a better one than EPOCHS' according to [5]. A potential drawback given in the same paper is that this package is not designed for power system simulations in particular. Thus, applying ADEVS to power system simulations may result in less reliable power system models and less scalable hybrid simulation.

GECO

The Global Event-Driven Co-Simulation Framework (GECO) is a power/communication network co-simulation framework that integrates a dynamic power system simulator and a communication network simulator using a precise timing mechanism. Here, the power system simulator is PSLF and the communication network simulator is NS-2. This framework is proposed in [1] and [33] where the authors announce it as an improved version of EPOCHS and ADEVS.

The improvement compared to EPOCHS is due to the fact that they use a *global event-driven* co-simulation framework [5]. This *global-event* synchronization method is more deeply explained in Section 2.4. Compared to ADEVS, they allow a faster co-simulation building while presenting results as accurate as ADEVS's. By contrast, according to [5], "the fidelity of the emulation cannot be guaranteed if the system is scaled down".

The framework structure can be seen in Figure 2.9, where the authors designed a bi-directional interface between the two simulators to allow interactions. In PSLF, a new dynamic model *epcmod* has been added as the main port to NS-2 and in NS-2, a new class *Tcl_PSLF* derived from TclObject has been designed in C++ as an interface to PSLF. Concerning the control strategies of the power system, they are designed in the *Power applications* classes and represent the functionalities of the software/hardware agents interacting with the power system and the ICT network in the real world [33].

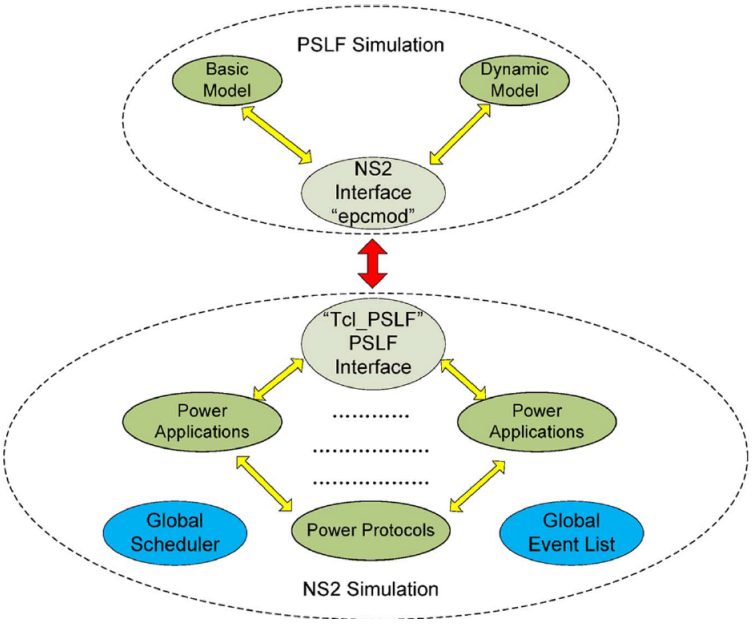


FIGURE 2.9: The structure of GECO's co-simulation framework [1]

VPNET

VPNET is a co-simulation framework that combines the Virtual Test Bed (VTB) for dynamic power system simulation and OPNET for communication network simulation. The idea of this framework is to study the interactions between advanced electrical systems and the communication networks allowing the monitoring and control of these electrical systems [6]. More precisely, it allows to analyze the impact of the performance of the communication network on the behavior, stability and safety of the power system and to study the influence of the energy demand on the design of the communication network. According to the developers, the VPNET approach allows for easier modification of network parameters (communication or power) during simulation compared to co-simulations based on code generation.

The structure of VPNET can be seen in FIGURE 2.10, where a co-simulation coordinator, implemented in C, allows data exchange and synchronization between the VTB and OPNET simulators. Note that this tool has been placed in this section despite the co-simulation coordinator because unlike the tools described in SECTION 2.3.2, the coordinator is not a pre-existing tool used to link simulators: it has been built for VPNET and is not available on its own.

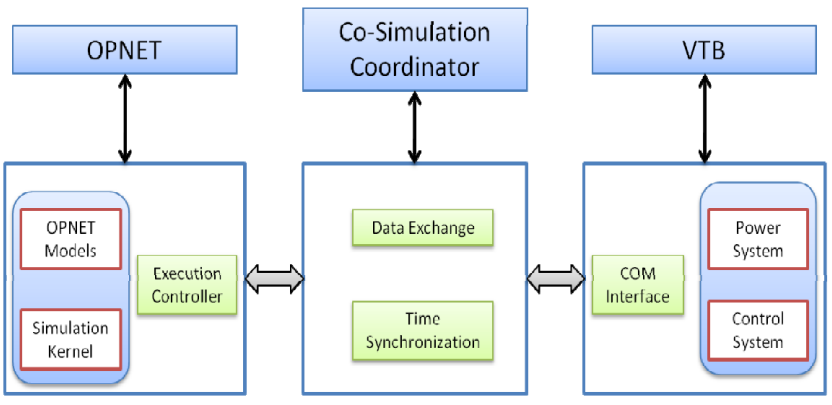


FIGURE 2.10: VPNET co-simulation framework. [6]

2.3.4 Choice

First of all, it has not yet been decided within the CYPRESS consortium whether the objective is to use a standardised architecture but if so, it should be noted that according to [49], using HLA and/or FMI to build a co-simulation adds a significant workload for developers, especially if the models of the co-simulation components often vary during the design process. However, there is a tool called CoHLA [92] which is an open source domain specific language (DSL) that allows the user to quickly construct a co-simulation from a set of simulation models. This is done using HLA (with the HLA implementation OpenRTI) for running the co-simulation and FMI for supporting simulation models created in different modelling tools. The aim of CoHLA is to minimise the effort, development and maintenance costs during the construction of a co-simulation [49].

Then, regarding the choice between using or not a master orchestrating the co-simulation, it should be noted that using a master middleware is a more generic solution than ad-hoc methods [63]. Indeed, it seems easier to add, change, remove specialised software (in the electrical or communication domain) with a central orchestrator without having to make major changes to the software in question. If one realises that a power system simulator is not capable of handling certain desired aspects and decides to replace it with another simulator, this should be easier with such an orchestrator component. Furthermore, ad-hoc developed tools are mostly developed for a certain purpose, for certain scenarios/scales and are therefore not specifically designed to achieve what is desired. For example, if one wants to simulate and analyse failures in the communication network, e.g. by injecting a cyber-attack, which is the case in the CYPRESS project, the ad-hoc developed tools are not necessarily designed to allow this.

However, if the aim is to create a co-simulation middleware from scratch instead of using an existing one, this should require as much effort and time than to perform an ad-hoc approach. Furthermore, it should be noted that using a master can potentially add delays/errors when running a co-simulation.

In the context of my Master's thesis, I decided to use an existing orchestrator to realize a small practical case. Indeed, my work being mainly focused on research, time does not allow to build a combination in an ad-hoc way or even to realize the whole co-simulation controller by myself, which would require significant efforts and a longer duration than the one attributed. Then, among the tools presented in this section, the choice fell on Mosaik. This choice is due to the fact that this framework is written in Python, which is the language with which I am the most comfortable (considering the time allocated to the practical part of my work, it would not have been feasible to learn a new language). Moreover, the tool used had to be compatible with my operating system, *i.e.* Windows. Finally, Mosaik is an up-to-date tool, well documented and the few modules present in its ecosystem are interesting and can be useful in the implementation of the practical case. Finally, even though mosaik is written in Python 3, its simulator API is compliant with each other language. Concerning the other options, it can be seen in Table 2.1 that they are written in other languages and Ptolemy II and FNCS have not been updated since 2018, which reinforces the choice of Mosaik.

2.4 Temporal synchronization

A very important challenge when building a tool combining several simulators whose simulations are driven in different ways (continuous time-driven or discrete event-driven) is the choice of the time synchronization strategy between these simulations during execution. The authors of [8] classify existing co-simulation platforms according to three main synchronization mechanisms: master-slave, time-stepped and global event-driven. This synchronization methods' classification is confirmed by authors of [7] and the mechanisms are described in the following subsections so that a choice can be made regarding them.
Note that these mechanisms present in the literature are presented here in order to have a global view on what exists and what is feasible, even if the choices made so far do not necessarily enable to use all possible synchronization techniques. In relation to these choices, SECTION 2.4.4 will also give the path followed for the synchronization in the practical part.

Also note that for simulators that use HLA: In addition to enabling the routing of messages between simulators and acting as an interface between them, the RTI of HLA-based frameworks also ensures synchronisation by ensuring that the simulation clocks remain synchronised between all components [3]. HLA-based tools seen until now use the time-stepped mechanism.

2.4.1 Master-slave

A first synchronisation mechanism is in the form of a master-slave configuration. One of the simulators in the co-simulation is designated as the master, whether it is the power system simulator or the communication network simulator (although it is often the discrete event simulator [7]). This master will have the highest priority during execution and will coordinate all the steps of the co-simulation [8],[7].

This method is represented in FIGURE 2.11, where during the whole simulation, the role of the master is played by the communication simulator while the power system simulator is the slave controlled by the master [8]. It can be seen that the master starts the simulation and runs as long as it does not need data from the slave (from t_0 to t_1 in the figure). Once it needs the data, the slave takes over and is simulated from t_0 to t_1 and then sends the information to the master [7].

The authors of [7] report the drawbacks of such a mechanism given in [2]. These are that the slave has to wait for the master's signal in order to communicate its events and that the execution is usually sequential. The fact that the slave has to wait the master induces that if a particular event occurs in a slave during its execution interval, it is necessary to wait for the synchronization point to warn the other simulators (and potentially the simulation controller, if there is one).
Note that according to [8], the master-slave method is simpler than the time-stepped and global event-driven methods described in the following subsections.

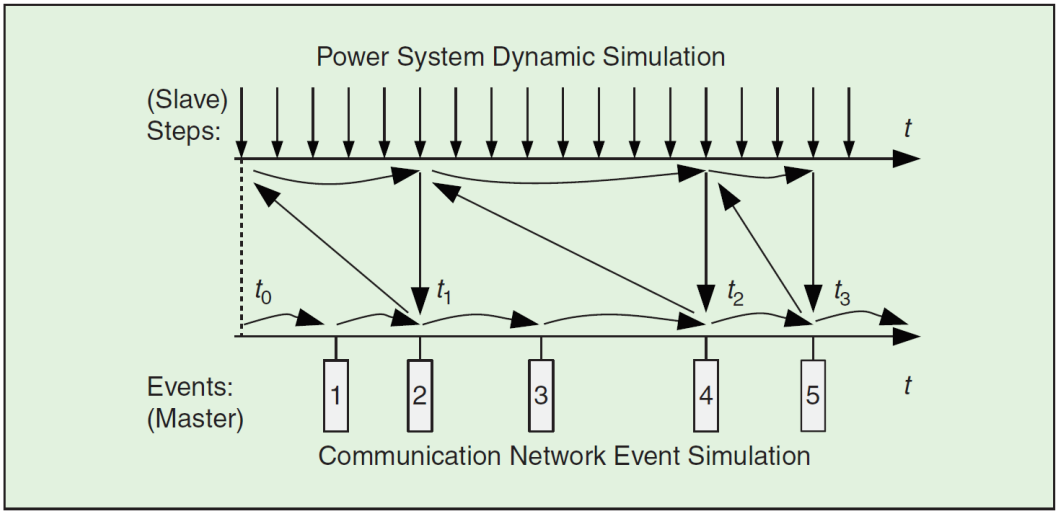


FIGURE 2.11: Master-slave synchronization method where the communication network simulator acts as the master. [7]

This method is used in the PowerNet co-simulator [93], which is the result of combining Modelica, an object-oriented simulation language offering an extensive set of standard libraries for complex system modelling [94], and NS-2 for network simulation. In PowerNet, Modelica is the slave and NS-2 is the master, which decides when the communication between the simulators will take place.
In this example implementation of the master-slave synchronization mechanism, they decided not to take the physical system simulator as the master because if it were, the inter-simulator communication could not depend on NS-2 events. Therefore, they preferred to keep the unpredictable and non-deterministic nature of the communication at the expense of the unpredictability of the physical systems. However, this is not a problem according to the developers because data sending from Modelica to NS-2 often consists of predictable events that can be coded beforehand and this is not often the case for data going from NS-2 to Modelica, especially because of possible delays or losses on the communication network [93].

2.4.2 Time-stepped or point based

A second method of synchronisation found in the literature is *time-stepped* or *point-based* synchronisation. It works as follows: as can be seen in FIGURE 2.12, each simulator runs its own simulation independently of the other simulators and stops at predefined synchronisation points in order to communicate with these other simulators [8]. In this case, the simulation tools run in parallel [7], so the components of the co-simulation must have this distributed capability.

One concern with this approach is that it can introduce imprecision. Indeed, if between two synchronisation points, a certain event takes place in a simulator and this event requires an interaction with another simulator, this event will be buffered in a cache waiting for the next synchronisation point. The problem is that this can lead to an accumulation of errors over time and make the simulation and its results wrong [8]. A solution proposed in several papers would be to move the synchronisation points closer together, e.g. by reproducing the same time steps as the dynamic power system simulation [7]. However, this solution could make the execution of the co-simulation very slow and become problematic in the case of large systems. Note that it is also possible to use dynamically assigned synchronisation points rather than predefined

ones, as was done in [95] where the next synchronisation point is a parameter of the power system simulator [7].

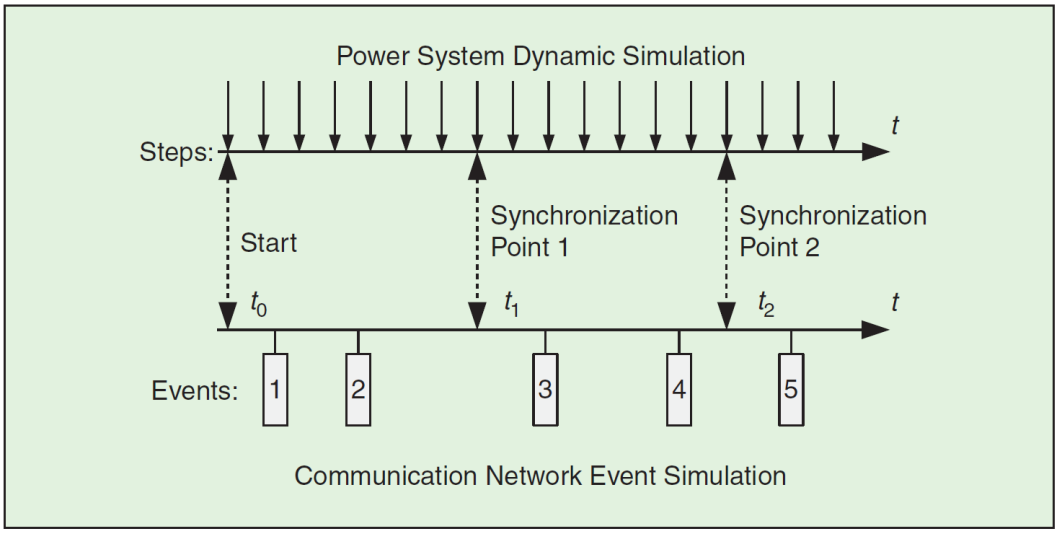


FIGURE 2.12: Time-stepped synchronization method. [7]

EPOCHS, which was introduced in SECTIONS 2.2.3 and 2.3.1, uses this synchronization technique, using pre-set points, which may effectively accumulate synchronization errors [40]. Indeed, the interaction between NS-2 and the continuous power simulators can lead to timing errors [3] and there may be a mismatch between the real and simulated dynamics and interaction [33]. This concern about time errors can be seen on FIGURE 2.13.

On this figure, we can see the temporal evolution of two simulators with above the evolution of a power system type simulation and below that of a discrete event simulator. We can see that there are predefined moments of synchronization represented by the red dotted lines at the moment when the 2 simulators will be able to exchange the necessary data such as the measurement of a voltage captured at the level of the power system and sent to the communication network simulator (to be then monitored by a control center for example). Errors occur when one simulator needs to interact with the other between two synchronization points, but this interaction must wait to be processed at the next synchronization point. For *Error 1* for instance, a certain event may have occurred in the power system simulation, such as the crossing of a current limit intensity, and this event requires a reaction at a certain component, via the communication network. For *Error 2* this can be due to the delay in communication: a variable, measured at *Synchronization Point 1*, is processed via the communication network in order to induce a reaction from a component on the power grid. However, due to the delay in the transport and processing of the information, the command is not communicated directly and once it is ready, it must wait for the next synchronization point.
All this is a problem because it induces undesirable delays that do not appear in a real system and can potentially accumulate over time [1].

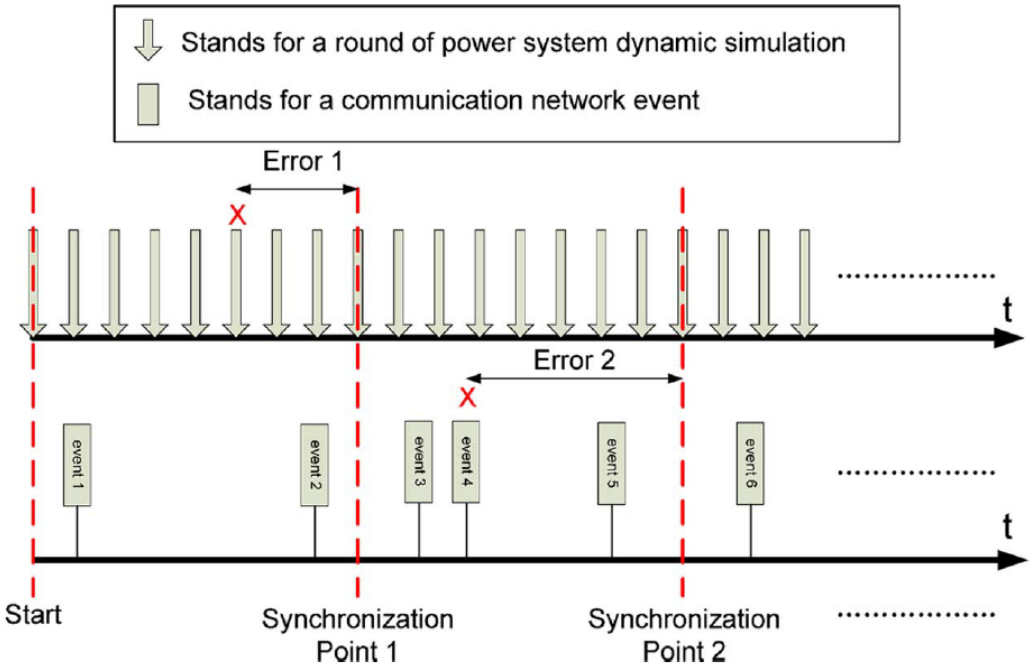


FIGURE 2.13: Time-stepped Synchronization method with accumulating errors. [1]

In [5], they qualify the synchronization technique of EPOCHS as being a trade-off between precision and efficiency and say that due to this synchronization technique, EPOCHS is not suitable for time-critical applications that require many interfaces between the power system and the communication network.

2.4.3 Global event-driven

A final synchronisation mechanism found in the literature is the (*global*) *event-driven* mechanism illustrated in FIGURE 2.14. In this mechanism, the iteration rounds of the continuous power system simulation are considered as discrete events and these are mixed with the events of the communication network in a *global event list*. This list forms an event queue and is ordered according to the timestamp of these different events [8]. In such an mechanism, the event processes must run one after the other [8] and a *global event scheduler* takes care of the queue by giving the hand to the simulators whose turn it is to run [7].

An advantage of this synchronisation approach, given in [5] is that the global event list managed by the global scheduler in order to arrange the event sequence ensures a synchronization error free framework, it removes the possible error accumulations that were discussed in the time-stepped mechanism [8].
A disadvantage raised by the authors of [8] is that the use of a global event scheduler relatively reduces the speed of co-simulation. This speed depends largely on the time step of the power system simulation but can also be increased by the interface between simulators, decreasing the scalability of tools implementing this synchronisation technique. In fact, performance is significantly impacted by the capabilities of the interfaces implemented [7].

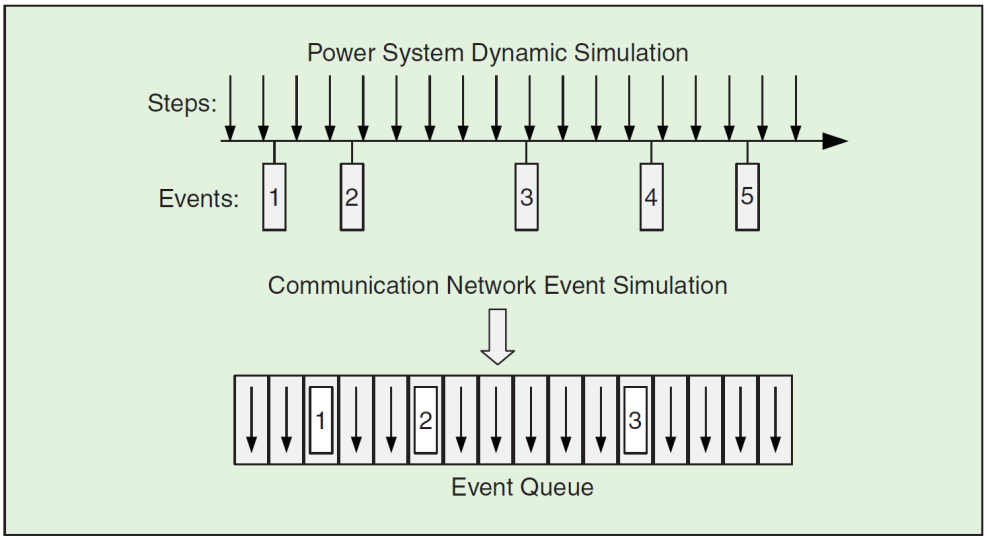


FIGURE 2.14: Global event-driven synchronization method [8]

GECO, which was introduced in SECTION 2.3.3, uses global event synchronisation [1],[33]. As was written above, it improves the time-stepped synchronisation mechanism. Indeed, the developers remove the explicit synchronization points found in EPOCHS and adopt a global scheduler for co-simulation. The global scheduler sees the dynamic simulator loops as its initialization events and, as can be seen in FIGURE 2.15, the simulation runs like this [33]:

1. The simulation starts: the global scheduler processes either a power system loop or a communication network event

2. Loop completed: the power simulator is suspended until the next cycle is processed by the global scheduler

⇒ Whenever there is a need for interaction between the two systems, the request can be processed immediately by the global scheduler without unnecessary waiting time

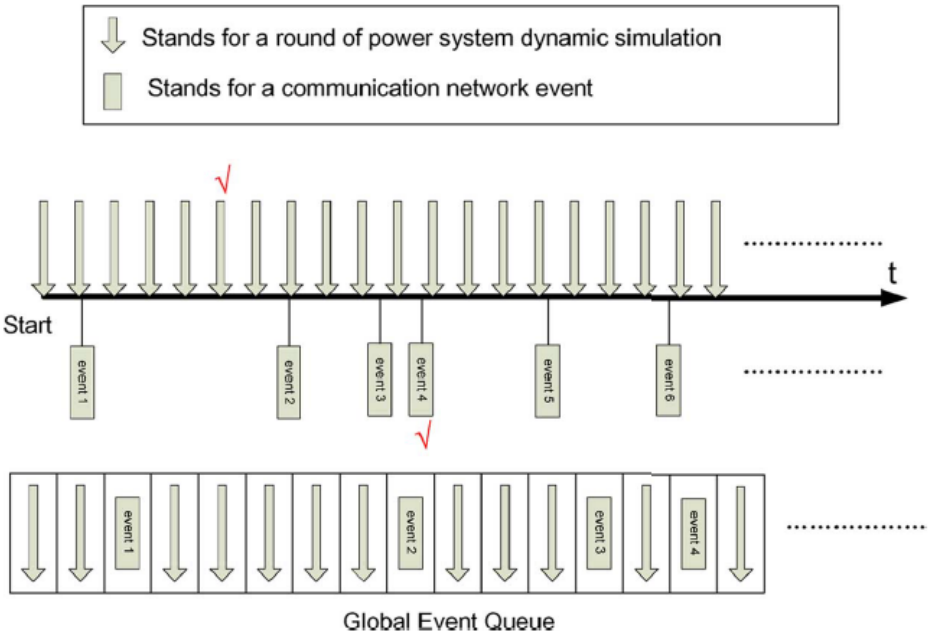


FIGURE 2.15: Co-Simulation framework adopting a global scheduler without synchronization errors [1]

Concerning the framework structure that could be seen in FIGURE 2.9, the integrated event scheduler of NS-2 is adopted as the global scheduler, and the global event-list is also derived from its counterpart in NS-2 [33].

2.4.4 Choice

Overall, these synchronisation techniques all have their advantages and drawbacks. However, the choice depends on the priority given to certain aspects: if the priority is to have precise simulation of fast power system dynamics, the time step used in power system simulation has to be small and in consequence, one suggests to use the time-stepped synchronization technique. On a performance point of view, this is much more recommended than employing the global event-driven technique in which the duration of the overall co-simulation would significantly increase as this synchronization technique would induce an enormous number of events in the power system simulation. However, the global event-driven method would have the benefit of reducing the probability of synchronization errors accumulation. Then, there is the master-slave method which is close to the functioning of the time-stepped one and is said to be simpler compared to the other two methods but which suffers from the same type of drawback as the time-stepped method with its issue concerning synchronization errors accumulation [8].

Note that the choice also depends on whether the platform and tools employed allow to run distributed simulations with different simulators running in parallel or only sequential execution. Even though the first case would be closer to the real operation of the smart grid, one must remain vigilant concerning synchronization since the simulators don't necessarily execute with the same speed. An interesting mechanism would be to have distributed co-simulation in which simulators stop and send signals as soon there is a need for interaction from one and/or the other simulator. As written before, the problem is that the simulators do not necessarily have the same rhythm, so a mechanism is needed to keep the same rhythm while running in

parallel.

As was written in the introduction of this section about temporal synchronization, the existing techniques in the literature have been presented but the choices made so far concerning the co-simulation approach and framework force anyway to take a certain premature decision. Indeed, by choosing to use mosaik, I chose to use a discrete event-based approach in which the step sizes of the simulators can be controlled during runtime. Indeed, a simulation with mosaik goes like this: all simulators start at time 0. When it is the turn of a simulator to perform its next step, it receives its current simulation time, then the step is performed and the simulator sends back to Mosaik the time of its next step, depending on the step size which may have varied with respect to a previous execution. To manage data exchanges between different simulators, Mosaik is based on the fact that a *simulator B* can run only if all input data are available. However, it is possible that data is made available at a certain time by *simulator A* but that the next step time of *simulator B* is later, at a moment when this data will have been overwritten by other data computed in the meantime by *simulator A*. In such a configuration, which can be seen in FIGURE 2.16 taken from the mosaik documentation [9], the step sizes must be chosen judiciously in order not to cause inaccuracies and temporal errors.

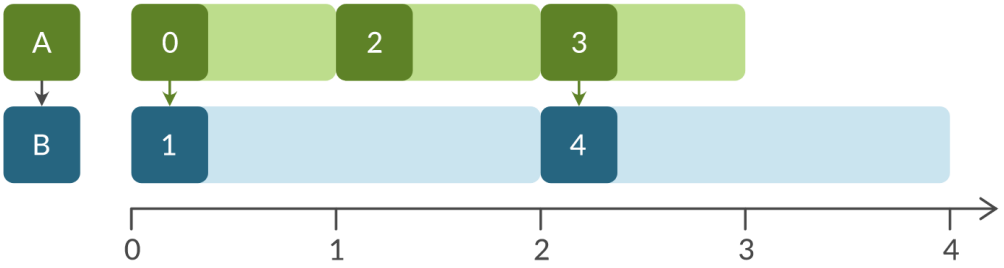


FIGURE 2.16: Data flow between two simulators in mosaik, with the data receiver that has a larger step size than the data sender. [9]

2.5 Specialized simulation tools

Now that the choices of coupling and synchronization methods between the communication network and power system simulators have been made, it is necessary to decide which simulators will be used to perform the co-simulation. Indeed, most tools developed in the scientific community use one or another specialized tool to perform the combined simulation. However, there is no indication that other simulators than those used cannot be used with the same type of architecture. This is by the way noted in [3] where they indicate that several tools could have been used to create EPOCHS. The choice is therefore made, beyond the framework approach, on the power and communication simulators. This section provides a list of the most popular power systems and communication networks simulators for this type of project.

2.5.1 Power system simulation

This section gives an overview of a broad list of tools specialized in power system simulation. In fact, in a power system simulator, the model used will depend on the choice of the developer, which in turn depends on the context and the targeted application [5].

As stated above, the methods of modelling and simulation of power systems can be broadly divided into two categories: steady state simulations and continuous time/transient dynamic simulations [5],[2].
The first category would come into play, for example, if one wanted to model the power system with a sequence of quasi-static power flow calculations. One can also estimate and balance the load or perform an optimal power flow study [2]. With steady-state methods, the steady state of the electrical system is computed with respect to (step-wise) constant inputs and boundary conditions. This state will correspond to a snapshot of this electrical system and a series of these snapshots will ensure a satisfactory simulation approach as long as we can assume that the system evolves in a quasi-static way. The system is thus analyzed in a stable equilibrium state and we make sure that the variables do not exceed the appropriate limits [2]. Note that the time steps in steady state simulations are usually expressed in seconds, minutes or hours. Finally, the calculation and analysis of the transient dynamics of electrical systems are not feasible with such methods, for this it is necessary to use the methods presented in what follows, *i.e.* the calculations in continuous time [5].
Indeed, if we want to take into account the transient effects during an EMT (Electromagnetic Transient) simulation for example, the continuous time calculations are more suitable. The oscillations due to a fault can then be analyzed or, more generally, the transitions between the equilibrium points induced by considerable changes in the configuration of the electrical system [2]. In this case, the physical equations related to the dynamic effects of the elements of the system are solved as a function of time, which induces very intense computational efforts. Sets of differential algebraic equations are mostly solved by numerical algorithms using discrete time steps, the size of which depends on the dynamics of the system. Here, the temporal granularity is finer than for the steady state, with time steps generally expressed in microseconds or milliseconds [5],[2].

In addition to this feature, simulators are often specialized in simulation of one (or several) of the hierarchical layers of the power system, *i.e.* generation, transmission or distribution. Based on these two aspects, a classification of several power simulators among which some are described in this section can be seen in Table 2.2. However, they are not all described since some of them offer nearly the same features as the described ones, then for those that are not described here, the interested user can refer to [2].

Finally, note that scalability of very large power system simulations is still (at least in 2015, when [5] was published) an important research topic that is just waiting to be solved. Overall, the performance of these dynamic simulations is still a research topic.

	Simulation Type		Power grid layer			Availability	
Simulator	Steady state (min, hours, days)	Transient Dynamics (s, ms, µs)	Generation	Transmission	Distribution + RCI loads	Commercial	Open Source
PSCAD/EMTDC [96]		X		X	X	X	
PSLF [97]	X	X	X	X		X	
DIgSILENT [98]	X	X	X	X	X	X	
PSS Sincal [99]	X	With add-On	X		X	X	License R&E
PSS E [100]	X	X	X	X		X	
EMTP-RV [101]		X		X	X	X	
PowerWorld [102]	X	With add-on		X		X	Educ. version
pandapower [103]	X		X	X	X		X
ETAP PSMS [104]	X	X	X	X	X	X	
Cymdist [105]	X		X		X	X	
EuroStag [106]	X	X	X	X	X	X	
OpenDSS [107]	X	X	X		X		X
ObjectStab [108]		X	X	X			X
GridLAB-D [109]	X			X	X		X

TABLE 2.2: Classification of power simulators inspired from TABLE 1 of [2].
RCI: Residential, Commercial and Industrial **R&E**: Research and education

PSCAD/EMTDC

EMTDC models short duration electrical power responses. PSCAD is a graphical interface used to simplify the development of EMTDC scenarios [3]. PSCAD/EMTDC is a continuous time system. EMTDC solves a series of differential equations using time steps in order to simulate power scenarios [110]. It contains very detailed electrical models and suits well to EMT analysis [3]. PSCAD is a commercial simulator but whose source code is not available [3].
This tool is used in EPOCHS [111] for transient protection simulation with short-term time domain responses [5]. It is also used in [112] to simulate power system control in a smart grid context. This tool offers the possibility to modify the length of the time step at will [3].
PSCAD/EMTDC is usable in HLA-based simulation, or at least with RTI since the software is used in EPOCHS. A PSCAD/EMTDC scenario can include user-defined libraries that add equipment definitions not present in the original software, using the C language [3]. Furthermore, the authors of [113] and those of [114] have coupled this simulator with MATLAB since it can be coupled with external tools [2].

PSLF

PSLF is a power system software that allows the simulation of steady-state and dynamic power systems [33]. It is broadly used by power suppliers to model electromechanical stability scenarios. As for EMTDC, this tool solves differential equations with help of times steps to simulate power systems [3]. PSLF is a commercial simulator but whose source code is not available neither [3].
It is used in EPOCHS for large-scale power system stability simulations [5] and is used in GECO too [33],[1]. PSLF is able to simulate a system as large as 125 000 buses [97]. It offers the possibility to change the time step length at will [3] and contains a rich library of electromechanical dynamic models [33].
As it was the case with EMTDC/PSCAD, this tool is usable with a RTI since the software is used in EPOCHS [3]. The software suite is written in Java and offers APIs to the user that might want to develop the product more deeply [33]. PSLF also comes with EPCL, a script language that allows for user-customized simulation setup and for adding new dynamic models to the model library [33]. However, according to [3], PSLF is more suitable for long scenarios than PSCAD as it models large system in less detail than the latter.

DIgSILENT - PowerFactory

PowerFactory is commercial a continuous time-based power system simulator [54] developed by DIgSILENT and that enables the modelling and analysis of power grids in the three hierarchical domains, *i.e.* generation, transmission and distribution, as well as in industrial domain [98]. This simulator numerically solves differential equations in a discrete time stepped manner [18]. This software has been used in INSPIRE, a HLA-based co-simulation environment whose goal is to analyze the real-time performance of wide area monitoring, protection and control applications [18],[54]. Since it is used as a federate in this HLA-based environment, it can be deduced that, by applying a few modifications, the simulator is compatible with this standard architecture. Indeed, DIgSILENT does not provide an interface to the HLA but by modifying its API, it is possible to connect, synchronise and manage its data objects within the HLA. In [18], the access to simulation data is done thanks to an OPC interface[9] and the authors of

[9]OPC is an interoperability standard for the exchange of data in some fields of the industry [115]. Note that OPC now stands for *Open Platform Communications* while it used to be the acronym for *Open Process Control*.

[116] also show that DIgSILENT offers the possibility to exchange power data with external software, since the simulator is combined with MATLAB using an OPC interface to exchange data between the tools.
In [40], DIgSILENT is used to simulate the power grid in a Hardware-In-the-Loop co-simulation. It is also used for power flow studies in [117] and transient studies in [118] since the software is able to simulate load flow, electromechanical RMS fluctuations and EMT. This way, it enables to perform studies going from transient network faults to longer-term power quality and control issues [2].
Note that thanks to the DIgSILENT Simulation Language, DSL, users have the possibility to create new models of electrical equipment [2].

Siemens' PSS® Product Suite tools

PSS® is a product suite that contains several software solutions such as PSS®SINCAL and PSS®E.

- **PSS®SINCAL**

PSS®SINCAL is a network planning and analysis tool that focuses on utility distribution system analysis. This tool is able to simulate several types of power engineering concepts, such as power flow, load balancing, load flow optimization and optimal branching. This is a commercial tool but that offers special licenses for research and education [2]. PSS®SINCAL offers a COM-server interface[10] for simplifying the integration into existing IT architectures [2]. In [119], the authors use the COM interfaces in Smart Grid simulation where PSS®SINCAL is used to analyze the impacts on integrating photovoltaic panels on the utility grid [2].

PSS®NETOMAC

PSS®NETOMAC is an additional core engine of PSS®SINCAL that enables advanced stability and transient analysis. In [120], they combined this simulator with NS-2 via an "external bridge" and the Java Native Interface (JNI) [18]. Furthermore, it could have been used in EPOCHS according to EPOCHS' developers [3].

- **PSS®E**

The PSS®E commercial tool targets transmission system planning and enables to perform load flow as well as transient analyses [2]. This simulator has been used in [121] for studying transient stability and in [122] for the control of power flow in transmission lines. According to their website, [100], PSS E has the advantages of saving time through scripting and automation via APIs and offering interoperability with other tools in the utility IT landscape. The user is able to interact with the tool by writing scripts in Python.

EMTP-RV

EMTP-RV is a commercial tool allowing to simulate electromagnetic and electromechanical transients in power systems [2]. EMTP-RV is used in [123], where the MV feeder response to indirect lightning strokes is modelled thanks to the software. According to the EMTP website [101], EMTP is "the most complete and technically advanced software for simulation and analysis of power systems. It is known to be the fastest, the most accurate and the most

[10]Component Object Model is a specification describing how an executable program can be packaged into an object by a programmer. In this way, the goal is to simplify the creation of programs by assembling different components [**?**].

numerically stable time-domain software in the industry". EMTP-RV could also have been used in EPOCHS according to the developers of EPOCHS [3]. Note that customized modules may be developed and we can interface them to the software through dynamic-link library functionality [2].

Pandapower

Pandapower is an open-source tool born from the combination of pandas, the renowned data analysis library, with PYPOWER, the power system analysis toolkit. It offers the possibility to perform power flow and optimal power flow studies using an improved solver as well as state estimation studies and short circuit calculations [10].
An advantage of pandapower is that it offers a very large library of models allowing the simulation of a very wide range of power system components. A comparison table with other open-source simulators, taken from the pandapower paper, is available in FIGURE 2.17. These components are defined by equivalent circuit models thoroughly validated against commercial software tools. Another advantage is that the user can modify and share this tool at will [103]. Finally, it should be noted that the development of pandapower was born out of the desire to bridge the gap between open-source and commercial tools. As written above it offers a large number of thoroughly validated and easily customizable templates, as do most commercial tools, and it also brings the flexibility and modifiability that open-source tools have [124].

	MATPOWER 6.0	PYPOWER 5.1.2	PSAT 2.1.10	OpenDSS 7.6.5	PyPSA 0.10	GridCal	GridLAB-D 3.2	**pandapower 1.4.3**
ZIP-load			✓	✓		✓	✓	✓
Line	✓	✓	✓	✓	✓	✓	✓	✓
2-Winding Transformer (π)	✓	✓	✓	✓	✓	✓	✓	✓
2-Winding Transformer (T)				✓	✓		✓	✓
3-Winding Transformer			✓	✓			✓	✓
DC Line	✓		✓	✓	✓		✓	✓
Ideal Switches								✓
Volt.-Controlled Generator	✓	✓	✓	✓	✓	✓	✓	✓
Static Load / Generation	✓	✓	✓	✓	✓	✓	✓	✓
Shunt	✓	✓	✓	✓	✓	✓	✓	✓
Asymmetrical Impedance								✓
Ward Equivalents								✓
Storage Unit				✓	✓		✓	

FIGURE 2.17: Comparison of open-source element mode libraries. [10]

ETAP PSMS

ETAP Power System Monitoring Simulation (PSMS™) is a commercial real-time power management system. According to ETAP's website [104], this software is capable of power management in several areas such as small and large electrical utility systems as well as generation plants, industrial sites or offshore oil platforms. It can perform a wide range of analyses such as load flow, short circuit, device coordination, transient stability and determine the appropriate response to a set of changes and disturbances occurring in the power system [2],[104]. The authors of [125] generate, with the help of the software, phasor data to which Principal Component Analysis is applied.

OpenDSS

OpenDSS is an open-source tool developed by the Electric Power Research Institute (EPRI) to simulate power systems with a particular focus on distribution systems. Its first role was to allow the analysis of electricity distribution planning with a focus on the interconnection between distributed generation and the utility system [2]. In addition, according to the EPRI website [107], this simulator can be used for almost all commonly used frequency domain analyses such as harmonic studies, volt-var control studies, etc. (on distribution networks) as well as for new types of analyses related to grid modernization, smart grids or renewable energy research.
In addition to the open-source aspect, an advantage of OpenDSS is that it is indefinitely extensible and therefore offers the possibility to be modified, with for example the possibility for users to define their own models [2],[107]. Moreover, this simulator allows the analysis of renewable energy sources that have stochastic and intermittent characteristics, which makes it a tool of choice in smart grid simulations. By the way, it is used in the smart grid co-simulation tool developed by the authors of [126] to study the influence of communication delays on the operation of the distribution network [2].
As written in Section 2.2.3, a use case presenting the control of the load tap of a transformer is realized in [43] using the Mosaik co-simulation platform and the developers used OpenDSS to realize the power part of this project. Another work using OpenDSS, coupled with OMNeT++ for communication, is SGsim [127], an open-source framework offering the possibility to simulate a variety of smart grid applications in real time. The impact of communications on power system control actions can be studied with SGsim but simulations of cyber-attacks are not possible [29].

Real-time hardware-based simulation

There exist also simulators that focus on real-time and hardware testing. Some examples are given but are not detailed here as they do not interest us in the current context.

eMEGASIM is a commercial power system (and power electronics) simulator for development and testing of control and protection equipment [128]. It is developed by OPAL-RT [129] and it enables real-time HIL simulations and rapid control prototyping (RCP). eMEGASIM offers an environment based on the Simulink tool [128].
Real Time Digital Simulator (RTDS) is a full digital power system simulator [130] and is the world's benchmark for real-time power system simulation [131].

2.5.2 Communication network simulation

To continue with specialized simulation tools, a shorter overview of the communication network simulators used in a smart grid context is provided in this section. They can also be found in Table 2.3 below.

Simulator	Language	Availability
NS-2	C++, Otcl	Open source
NS-3	C++, Python	Open source
OMNeT++	C++	Free for academic and non-profit use
NetSim	Java	Commercial for use at the undergraduate level
NeSSi^2	Java	Open source
OPNET Modeler	C (C++)	Commercial
QualNet	C++	Commercial

Table 2.3: Classification of communication network simulators inspired from [14].

Network Simulator (NS-2 and NS-3)

• **NS-2**

NS-2 is an event-driven communication network simulator that allows the creation of a wide variety of communication scenarios (e.g., a scenario where each relay/circuit breaker location is represented as a communication node). This tool is able to simulate TCP and UDP behavior under several forms of stress [3]. NS-2 is able to simulate four layers of the OSI model for computer network protocols (application, transport, network and data link layers). The simulation tool also allows simulations of wired, wireless and hybrid communication [33]. Ns-2 is written in C++ and an Object oriented version of Tcl called OTcl [132].
The code of NS-2 is open-source and the simulator is used, among others, in EPOCHS [3], in [120], in GECO [33],[1].
According to [5], able to simulate a network at least 20 000 nodes. It is possible to add events to NS-2 to ensure that it will use a fixed length between each time step (like power simulators) [3]. In [3] (2006), they said its use is relevant to power systems because industry standards for future generations of power electronic equipment would use TCP over Ethernet networks. However, it should be noted that NS-2 is old and has not been updated since 2011 [133].
NS-2 is usable in HLA-based simulation, or at least with RTI since the software is used in EPOCHS. It is also possible to add new communication protocols in C++ to NS-2 and transport protocols to serve as a link to an RTI for example, which is done in EPOCHS [3]. Furthermore, in GECO [1], NS-2's integrated network event scheduler shows its ability to act as a global scheduler in a combined simulation.

• **NS-3**

NS-3 is a discrete-event computer network simulator, this is an upgraded version of NS-2 rebuilt from scratch [134]. It is open-source too. However, NS-3 was not used in "old" cyber-physical combined simulations but some papers using it could be found from approximately 2014. In [135], they build a co-simulation framework to analyze the performance of power and network heterogeneous system. For that, they integrate MATLAB for power system to NS-3. In [76],

they introduce FNCS, a federated approach allowing co-simulation of both transmission and distribution level power grid simulators with the communication network simulator. NS-3 is integrated with GridLAB-D (distribution) and PowerFlow (transmission). In [136], they introduce FSKIT, a federated simulation toolkit coupling continuous time and discrete event simulations to perform power/communication co-simulation. They do that by coupling GridDyn (a High Performance Computing oriented power system dynamic simulator) to NS-3.
While NS-2 was based on the use of scripting languages for simulation control (Tcl/Tk) and only the core of the simulations was implemented in C++, NS-3 is written entirely in C++ with optional Python bindings, which allows simulation scripts to be written in C++ or Python [137]. Development of NS-3 is still ongoing, meaning that it is still enhancing.
Note that NS-3 is not backwards compatible with NS-2, so that NS-2 simulation models must be implemented gain for NS-3 [2]. But according to the abstracts of [76] and [136], NS-3 seems to be usable in a federation like it was the case for NS-2.

OMNeT++

OMNeT++ is a discrete event simulation environment that is primarily used to simulate communication networks, both wired and wireless, but its general design also offers the possibility of simulating other distributed systems in general. Simple modules constitute the simulation models and compound modules, defined thanks to OMNeT++'s own language (Network Description Language), consist of other simple or compound modules. Regarding the protocols supported by the environment, the INET framework extension provides the ability to simulate Internet-based protocols such as IPv4, TCP, UDP and Ethernet, among others [2]. OMNeT++ also provides the ability to perform parallel simulations and thus facilitates the simulation of large networks.
Note that like NS-3, OMNeT++ is still being developed and improved, with the latest version dating from April 2021 [138]. In addition, although it is an open-source environment [139], OMNeT++ also includes a commercial version called OMNEST, which provides support for the HLA standard [140],[2].
An intelligent network is simulated in [34] where OMNeT++ acts as a development platform. In [91], the authors integrate OMNeT++ with ADEVS-THYME. Overall, this simulator has been widely used for the development of smart grid simulators [2].

NetSim

NetSim is a network simulator and emulator supporting a wide range of technologies and allowing to simulate Cisco Systems[11] networking hardware and software. It is a discrete event simulator with an object-oriented environment to simulate voice and data communication scenarios for High Frequency Global Communication Systems (HFGCS) [14].
NetSim is commonly used to simulate the communication layer of power networks [142] and is a tool that continues to be improved today, as its latest version was released in May 2021. The tool is available in a standard version, a professional version and an academic version [143].

[11] Cisco Systems is the world leader in the design, development and marketing of networking equipment for the Internet [141].

NeSSi

NeSSi² (Network Security Simulator) is an open-source discrete event simulation framework. It simulates communication networks and is mainly dedicated to the security of these networks [144], offering the possibility, among others, to model attacks (which can be automatically generated based on profiles), to detect them (by analyzing the traffic and via detection algorithm plugins) or to evaluate security countermeasures [29]. NeSSi² is developed on the JIAC service agent framework, which brings the feature of a distributed and easily extensible architecture [29].
An advantage, as it was the case for other network simulators, is that one can realize a distributed simulation and thus study large-scale networks [2]. According to the software's website, NeSSi has already proven useful in testing intrusion detection algorithms as well as in network security analyses.
However, one drawback with this simulator is that the current version, *i.e.* the latest version to be developed, was released in 2013, which means that NeSSi² has not been improved for a long time. In the field of smart grids, NeSSi² has been used in an integrated approach dedicated to the optimization of an agent-based smart grid management system [145] as well as in a federated simulation to analyze the security of smart metering [146].

OPNET Modeler

OPNET Modeler is a proprietary software tool for modelling and simulation of communication networks based on discrete event simulation. It offers built-in validated models such as LTE, WIMAX, Wi-Fi, etc. It is a powerful tool capable of taking into account path loss and mobility of communication network models [54],[2].
OPNET Modeler has already been combined multiple times with power system simulation in co-simulations, helped by its open interface offering the possibility to integrate other simulators. The addition of external object files, libraries or hardware is also possible through this interface [2]. As already written above, in INSPIRE [147],[54] they use HLA as a co-simulation standard combining DIgSILENT PowerFactory and OPNET Modeler, which means that the latter can be used in a simulation based on HLA. It provides an HLA interface to connect to an HLA federation according to INSPIRE authors. In addition, the authors of [3] suggest that it could have been used in EPOCHS.
Finally, OPNET is also used in the TASSCS project (A testbed for analyzing security of SCADA control systems) in which a testbed is created dedicated to study and simulate the methods available for securing and protecting SCADA systems against cyber attacks [148]. In this project OPNET Modeler is linked to PowerWorld and hardware is integrated, and in the paper [148] they present results regarding the detection of some cyber attacks and the evaluation of protection techniques, showing that the testbed can help to identify and reduce the consequences of compromised HMI and DoS attacks [29].

QualNET

QualNet is a real-time commercial simulator that allows for the planning, testing and training of communication networks. It is able to mimic the behavior of a physical communication network with very high fidelity and provides numerous wired and wireless network models and protocol models for the analysis of wired, wireless and hybrid networks [149].
This simulator is capable of supporting large-scale heterogeneous networks (thousands of mobile nodes) as well as distributed applications running on these networks. The software can

effectively run in a distributed way thanks to the use of PARSEC, the parallel simulation environment for complex systems [14].
QualNet was used in [40] where they build a Hardware-In-the-Loop cyber-physical co-simulation platform for power system and simulate a small cyber-attack on it. Indeed, as QualNet is a real-time simulator, some communication hardware can be connected to it and form a HIL simulation system with this external real network linked to a virtual network built in the simulator.

2.5.3 Choice

Regarding power system simulators, a choice can be based on the characteristics displayed in TABLE 2.2, *i.e.* the simulation type available and power grid domain simulated with the respective tools as well as the license (proprietary or open-source tool, with educational version or not,...).
For the CYPRESS project, the desired type of simulation has not been specified, but in order to perform the most comprehensive work possible, allowing for a wide variety of cyber-physical tests and scenarios (e.g. cyber-attack scenarios), it is most likely best to choose a tool that allows for both steady-state and transient dynamics simulations. In other words, the two main categories of simulation, the two types of "time scales". Indeed, the degradation of the operation of the cyber layer of the smart grid can have a lot of different consequences for which it is better to have the most complete tool with the most modelling capabilities possible.
Therefore, tools that only simulate transients or that only handle steady state are less interesting, unless one assumes that several power system tools can be incorporated and used, in which case two simulators could be chosen.

Furthermore, since CYPRESS will focus on the bulk power system, simulators specialized in the transmission network are welcome but having the capability to simulate also the other parts of the power grid is obviously a good point as the impacts of faults appearing on the ICT layer may expand and appear anywhere in the grid.
A final criterion, the importance of which is not clearly specified is the license nature of the tool. However, it seems that open-source tools will be preferred, according to a member of the CYPRESS project team.

About the communication network simulation tools, they are less varied and seem to present quite similar features from one to the others. Once again, the choice may depend on the license-free character of the respective tools. Network Simulator seems to be a good choice, with the more recent version NS-3 even though NS-2 was used in most of the old reference works. Anyway, most of these tools are all widely used and adopted in industry and/or research so that others seem to be usable as well, not knowing exactly what protocols and other details will be needed for this ICT layer in the project.

To conclude with the project, since it has not yet been specified what exactly will have to be modelled, I have resumed what exists and after that it will depend on the choice towards which the project will go.

Concerning the practical part of my Master's thesis, I chose to use the power systems simulation library pandapower because this one, as noted in SECTION 2.5.1, has many advantages while being totally open-source. Moreover, it is used with Python and pandas is a very well known and used library which can be trusted. Moreover, an advantage compared to the short

time available to realize the practical part is that pandapower is already part of the mosaik ecosystem, offering an API implemented to suit mosaik and allowing to use directly this tool with this co-simulation environment.
For the communication network simulator, it was decided to also use a tool already adapted to mosaik, *i.e.* the *mosaik-communication* suite which allows to simulate the behaviour of communication networks in a rather simplistic way. The choice was initially made to use NS-3 which, as explained in Section 4.2, is a real reference in its field and is also integrable to mosaik. However, the work done for this master thesis was mainly a research work dedicated to provide knowledge and information about the possibilities of co-simulation for the CYPRESS project. Therefore, the short time allocated to the practical part did not allow the realisation of the API necessary for the integration of NS-3 to mosaik. Nevertheless, this integration is feasible and the project team will most probably realize it in a future development.

2.6 Conclusion

To conclude this broad chapter reviewing the state of the art of the tools available to combine the cyber and physical parts of cyber-physical power systems within a simulation, the different choices are briefly recalled.

To begin with, it has been seen that there are challenges in combining modelling and simulation tools as different as those for power systems and ICT networks. The main challenge is the temporal aspect, with continuous simulation on the one hand (although very often performed in periodic time steps) and discrete event simulators on the other hand, where events can appear at any time.

Then, the most general types of approach to achieve the combination of these two domains within a simulation were discussed, with mainly the choices of a single environment integrating the 2 domains (by modelling one of the two within the other) or that of a co-simulation combining tools specialised in their respective domain. The co-simulation was decided upon because it is by far the most represented in the literature. Furthermore, the first solution seems limited in its ability to perform complex simulations and integrating one domain into a simulator for the other domain seems complicated to implement in practice.

After this, there was a section dedicated to the existing approaches to implement a co-simulation in which the final choice fell on co-simulation with orchestrator controlling different tools, synchronising them and allowing the exchange of their data. This is the solution that will allow the most genericity unlike ad-hoc approaches which are very often specific to the cases for which they were created.

The different possible methods of synchronisation between the simulators of a co-simulation were then discussed. They all have advantages and disadvantages and a proposal for a possible mechanism was given along with an explanation for the synchronisation within mosaik, which is the co-simulation orchestrator that will be used here.

Finally, once all the choices and possibilities have been made, it was time to see which power system and communication network simulators to add to the co-simulation. The choice is very broad but for the next chapter it was decided to make a co-simulation combining pandapower and the communication suite *mosaik-communication* around mosaik.

The following chapter will therefore be a kind of proof of concept attesting to this combination with a practical use case and a test scenario to demonstrate how it works.

Chapter 3

Co-simulation platform implementation and test case

Once the state of the art of existing co-simulation platforms was completed and choices were made, the practical implementation of a platform combining these choices could be initiated. The aim of this chapter is twofold. Firstly, it aims to provide a sort of proof of concept of the choices made, to show that mosaik does indeed work and correctly manages synchronisation, data exchanges between simulators, etc. Secondly, the objective is also to illustrate the impact that communication network failures can have on the operation of the power grid. This is done with the help of a small and common test case.

As a reminder, the tools used are: mosaik as a co-simulation framework handling the synchronisation of simulations and data exchange between simulators. Pandapower to carry out simulations of power flow calculations for the electrical network. For pandapower, a package containing an adapter to connect pandapower to mosaik called *mosaik-pandapower* was used. Finally, the cyber layer of the cyber-physical use case was modelled and simulated using the *mosaik-communication* suite also adapted to mosaik. As explained in Section 2.5.3, this package allows a simplistic simulation of the communication network and was preferred to NS-3 for the small practical case implemented here.

This chapter presents the different steps necessary to build the co-simulation. To begin with, a use case was defined and is described in Section 3.1. Then, in order to get to grips with pandapower, a first test of the use case was first carried out using pandapower alone in Section 3.2. Then, after using pandapower to generate an input file that will be useful in mosaik, the explanation on how combination of mosaik with pandapower (and a variable load simulation module) was built is given in Section 3.3. To continue, control was added to the resulting co-simulation before integrating communication with all these components. These steps are explained in Sections 3.4 and 3.5. Finally, Section 3.6 concludes this chapter with a test scenario highlighting the impact that a communication network failure can have on the operation of the cyber-physical power system.

3.1 Use case description

To illustrate the implementation of co-simulation and the combined use of mosaik, pandapower and the communication suite *mosaik-communication*, a small test case has been realized, which is described in this section. To start with, the topology of the small electrical network is

described, before talking about the implemented control system highlighting the ICT layer of the final system.

3.1.1 Power grid

The power grid considered is simply the interconnection that we could have between a transmission network and a distribution network. The idea was to do as in a substation with a bus connected to the transmission network from where the power flow would arrive, followed by a transformer with an on-load tap changer (whose principle and control are explained in the next section). The secondary of the transformer would be connected to a MV bus, which would be connected via a line to a bus connected to the distribution network. On this line between the transformer's MV side and the distribution bus, another bus would be added in order to allow the connection of arbitrary elements and/or to perform measurements to know the state of the system at this point.

To summarize, the network to be modelled contains the following elements that can be observed in FIGURE 3.1:

- `feeder_bus` the bus representing the feed from the external transmission grid;
- `tfo_hv_mv` the transformer equipped with on-load tap changer;
- `tfo_lv_bus` the bus connected to the secondary of the transformer;
- `meas_bus` the bus at wich some measurements can be made or other elements added;
- `dist_bus` the bus connected to the distribution system (load);
- and finally, the lines connecting `tfo_lv_bus-meas_bus` and `meas_bus-dist_bus`.

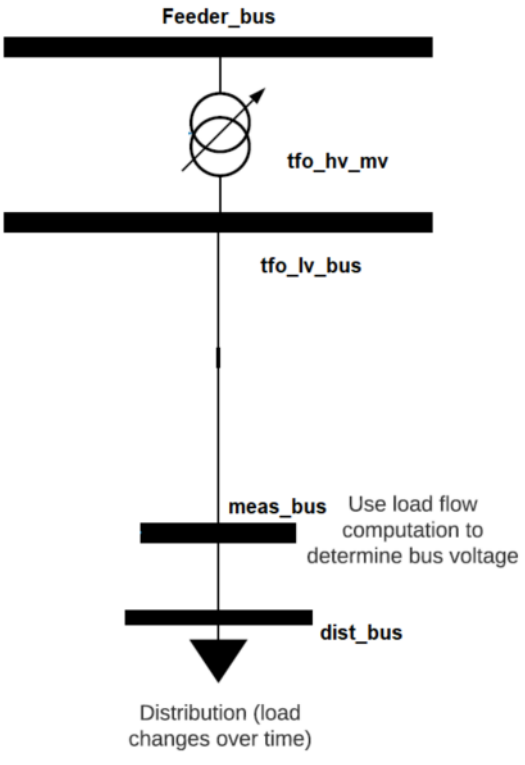


FIGURE 3.1: Power grid topology of the considered use case

3.1.2 Control and communication

The goal being to see the impacts that the coupling between the electrical network and the communication network can have, a control requiring the communication has been added to the power grid system. Indeed, as written in the previous section, the transformer is provided with an on-load tap changer.

An on-load tap changer is a mechanism that will modify the tap position of the transformer, which will modify the number of turns (and thus the turn ratio) with the final goal of adjusting the voltage in the vicinity of the transformer. The principle is illustrated in a simplified way in FIGURE 3.2, where it can be seen that the windings of the primary (on the left) can vary.
The on-load tap changer has the particularity to change the windings without interrupting the current. It can be controlled either manually, remotely by the control center operator, or automatically with a local control system. Here it is the automatic control mechanism that will be implemented. Note that in general, this is the windings on the side with the highest rated voltage that are modified because it is the side with the lowest currents and the most winding turns. This is therefore on this side that the tap changer has been placed in the use case [11].

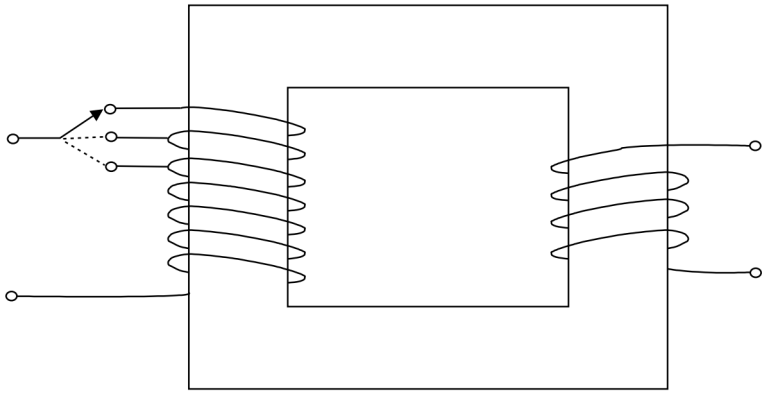


FIGURE 3.2: Tap changer principle. [11]

Still on the network of FIGURE 3.1, the idea was to control the tap of the transformer on the basis of voltage measurements made further away, near the distribution, *i.e.* at the `meas_bus`. For the measurement we would have a merging unit, which is a device that collects the instantaneous values of current and voltage, samples them and sends them to the protection and control units. Note that these instantaneous values are taken at the current and potential transformers, instruments that put down the current and voltage to a safe limit value that can be easily measured by the measuring instruments [150],[151].
Then, the measurement would be transported through a communication link to an intelligent electronic device (IED) in which the on-load tap changer control would be implemented. Finally, once the measurement has been processed and the calculations made, the tap position command would be sent to an actuator at the transformer's primary via a second communication link.

To summarize, the cyber-physical system can be seen in FIGURE 3.3, where the following control and communication elements have been added:

- `MU` the merging unit collecting voltage value from `meas_bus`;

- `OLTC_controller` the IED receiving voltage value, computing a tap position and sending it to its output port;
- `tap_actuator` the communication node acting on the tap changer of the transformer;
- and finally, the communication links connecting `MU-OLTC_controller` and `OLTC_controller-tap_actuator`.

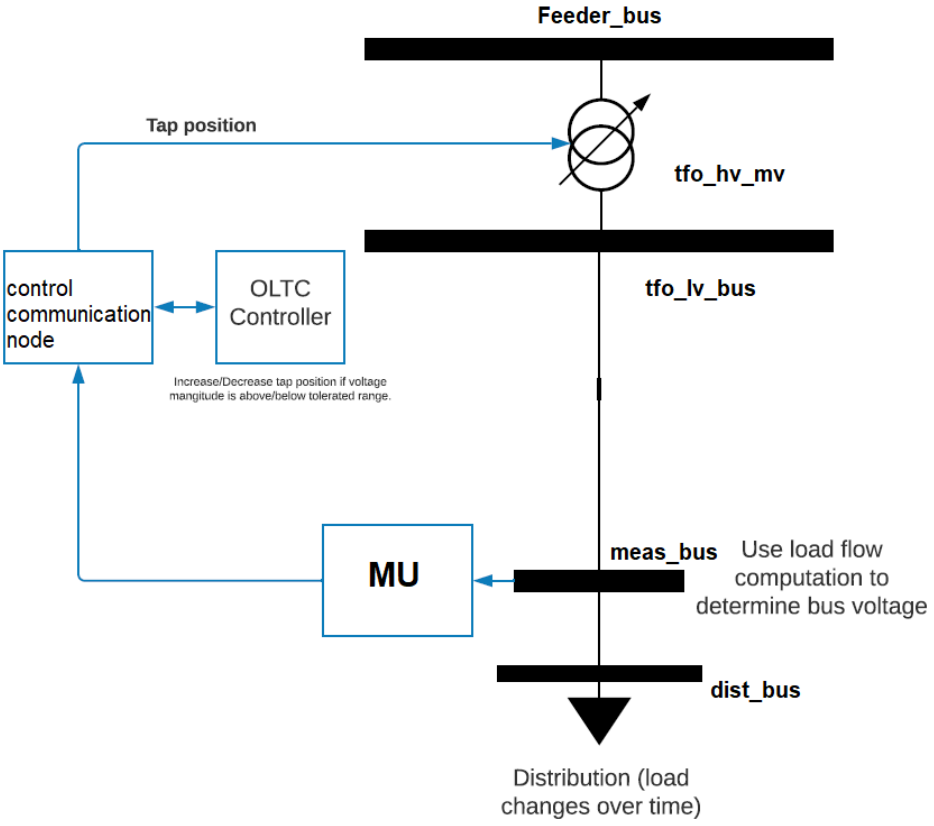


FIGURE 3.3: Cyber-physical use case topology.

3.2 Getting started and first tests with pandapower

Once the choice of the topology of the cyber-physical system and the elements constituting it were made, the construction of this one in order to realize the co-simulation began. To do so, it was first necessary to familiarize with the package pandapower by realizing a simulation which will be useful later during the integration to mosaik. This section deals with this first step in the design process of the practical use case and starts by presenting how the electrical elements are modelled in pandapower.

3.2.1 Equivalent circuit models

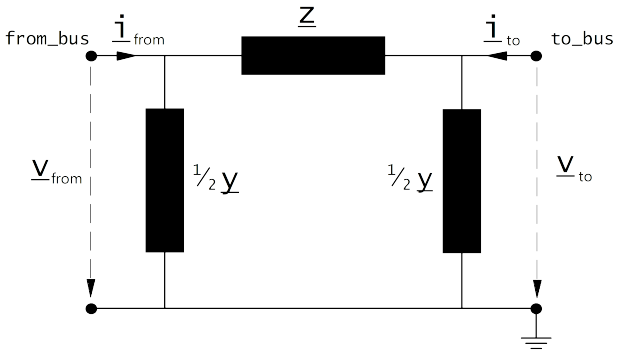
To start with it is important to know how the different elements chosen for the electrical system, *i.e.* bus, line, transformer (with tap), load and external grid (transmission),are modelled within pandapower. This subsection briefly describes the electrical models used for these elements based on the pandapower documentation [12].

Bus

A bus is simply modelled as a node in the network characterized by a nominal voltage $V_{n,bus}$ (kV) and whose result after execution of a power flow are the voltage magnitude $|\underline{V}_{bus}|$ in per unit, the voltage angle $\angle\underline{V}_{bus}$ [degree] and the active and reactive power demands $\mathrm{Re}\left(\sum_{n=1}^{N}\underline{S}_{bus,n}\right)$ [MW] and $\mathrm{Im}\left(\sum_{n=1}^{N}\underline{S}_{bus,n}\right)$ [Mvar] from this bus (the power is negative if the bus provides power because the receiver convention is adopted).

Line

A line is modelled with a π-equivalent circuit, which is the most commonly used model and that can be seen in the following figure next to the relations to calculate the elements of this equivalent circuit. In these relations, L [km] is the line length, r [Ω/km] the line resistance per unit length, x [Ω/km] the line inductance per unit length, g the line conductance per unit length [µS/km], c [nF/km] the line capacitance per unit length and finally $n_{//}$ the number of parallel lines if there are some.



$$\underline{z} = (r + j \cdot x) \cdot \frac{L}{n_{//}}$$

$$\underline{y} = (g \cdot 10^{-6} + j \cdot 2\pi f \cdot c \cdot 10^{-9}) \cdot L \cdot n_{//}$$

FIGURE 3.4: Line π-equivalent model. [12]

Transformer

For transformers, pandapower provides two equivalent circuit models, the T-model or the π-model. The T-model, whose drawing is given here-below, was chosen for the use case. The parameters that are needed to use a transformer in a power flow calculation are the rated apparent power of the transformer, the rated voltage at both sides, the short circuit voltage, its real component, the iron losses and the open loop losses.

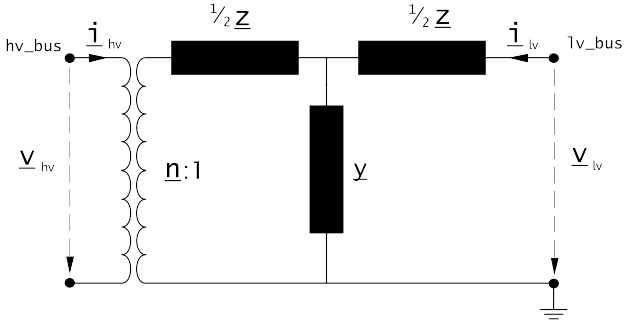


FIGURE 3.5: Transformer T-equivalent model. [12]

The transformer ratio is given by

$$\underline{n} = n \cdot e^{j \cdot \theta \cdot \frac{\pi}{180}}$$

with transformer phase shift angle θ and magnitude

$$n = \frac{V_{n,HV,transformer}}{V_{n,LV,transformer}} \cdot \frac{V_{n,LV,bus}}{V_{n,HV,bus}}$$

where $V_{n,HV,transformer}$ and $V_{n,LV,transformer}$ are the rated voltage on high voltage and low voltage sides of the transformer while $V_{n,HV,bus}$ and $V_{n,LV,bus}$ are those inherent to the buses attached to the transformer.

The effect of the load tap changer on the turn ratio is the following:

$$n_{tap} = 1 + (tap_{pos} - tap_{neutral}) \cdot \frac{tap_{step,percent}}{100}$$

where tap_{pos} is the current position of the tap, $tap_{neutral}$ the rated tap position and $tap_{step,percent}$ [%] the tap step size for voltage magnitude. In consequence, since the tap changer is placed on the HV side as written in SECTION 3.1.2, the reference voltage of the HV side $V_{n,HV,transformer}$ is multiplied by n_{tap}.

Load

A load must be attached to a certain bus created beforehand and when performing a power flow calculation, the loads are modelled as PQ buses to which an active power P_{load} [MW] and a reactive power Q_{load} [Mvar] are assigned. These powers, as written above, are given in the receiver convention, meaning that P_{load} must always be positive.

External grid

The last element that is part of the considered use case is an external grid. In pandapower, these elements are created to represent a connection to an higher voltage level of the grid. Here it will correspond to a connection to the transmission grid reaching a power substation. In power flow calculations, buses attached to external grids are modelled as slack buses (buses with a fixed voltage amplitude, usually 1 p.u., and whose voltage angle $\theta = 0°$ serves as angular reference for other buses) given that the external grid is modelled as a voltage source.

Note that unlike the case of loads, the convention used for external grids is the generator convention, which means that P_{ext} is positive if the external grid supplies power, negative if it absorbs power.

3.2.2 Test system implementation in pandapower

Having seen the electrical models representing the elements of the system, it was time to create the system in pandapower.
To begin with, it was necessary to create an empty network, the parameters of which were chosen to be the frequency of 50 Hz and the nominal apparent power of 100 MVA, useful for the per unit calculations.

```
net = pp.create_empty_network(f_hz=50.0, sn_mva=100)
```

Then the various buses were added to this empty network:

```
# Feeder (= transmission system = external grid)
feeder_bus = pp.create_bus(net, name='feeder_bus', vn_kv=220.)
# Secondary bus of the transfo (LV side)
tfo_lv_bus = pp.create_bus(net, name='tfo_lv_bus', vn_kv=110.)
# Bus from which measurements will be done
meas_bus = pp.create_bus(net, name='meas_bus', vn_kv=110.)
# Bus for the distribution (load)
dist_bus = pp.create_bus(net, name='dist_bus', vn_kv=110.)
```

The lines were then added, creating the links between the buses in the medium voltage section:

```
# Line from secondary of tfo to measurement bus
line1 =pp.create_line(net, from_bus=tfo_lv_bus, to_bus=meas_bus,
                      length_km=0.1, std_type="122-AL1/20-ST1A 110.0")
# Line from measurement bus to distribution bus
line2 = pp.create_line(net, from_bus=meas_bus, to_bus=dist_bus,
                       length_km=0.1, std_type="122-AL1/20-ST1A 110.0")
```

As can be seen from these code lines, the length of the two lines is fixed at 0.1 km and the various parameters required (resistance per unit length, inductance per unit length,...) are included in the `std_type` field, where the type `"122-AL1/20-ST1A 110.0"` is a standard line type. Indeed, even though the user can specify the different parameters himself, pandapower offers a library of standard default types to be used more rapidly. The parameters of the chosen line can be found in the following table:

Line `122-AL1/20-ST1A 110.0`			
Resistance of the line [\Ohm/km]	0.2376	**Maximal thermal current** [kA]	0.41
Inductance of the line [\Ohm/km]	0.43	**Line type** (overhead line or underground cable)	overhead line
Capacitance of the line [nF/km]	8.5	**Conductor cross-section** [mm²]	122

TABLE 3.1: 122-AL1/20-ST1A 110.0 line parameters. [12]

After that, the external grid was attached to bus `feeder_bus` and the transformer was placed between the latter and bus `tfo_lv_bus`:

```
# External grid
pp.create_ext_grid(net, bus=feeder_bus)
# Transformer
pp.create_transformer(net, hv_bus=feeder_bus, lv_bus=tfo_lv_bus,
                      std_type="100 MVA 220/110 kV")
```

This transformer was created using a pre-existing standard from the pandapower library, as was done for the lines. The parameters for the type of transformer used are given in the following table:

Transformer `100 MVA 220/110 kV`			
Rated apparent power [MVA]	100	**Rated voltage at high voltage bus** [kV]	220
Rated voltage at low voltage bus [kV]	110	**Short circuit voltage** [%]	12
Real component of short circuit voltage [%]	0.26	**Iron losses** [kW]	55
Open loop losses [%]	0.06	**Transformer phase shift angle** [°]	0
Rated tap position	0	**Minimum tap position**	-9
Maximum tap position	9	**Tap step size for voltage magnitude** [%]	1.5

TABLE 3.2: 100 MVA 220/110 kV transformer parameters. [12]

Finally, a load was added to `dist_bus`:

```
# Load
pp.create_load(net, name='dist_bus_load', bus=dist_bus, p_mw=100.)
```

With this single line of code, a constant load of 100 MW is installed at the distribution bus. Nevertheless, as the aim was to vary this load to observe the operation of the tap changer, some manipulations were carried out in the code, and a tap changer control was even tested to attest to the capabilities of pandapower before integrating it into the co-simulation.
However, these manipulations are not detailed here because they were put aside in the following steps. Indeed, they did not involve communication and were not useful for the implementation of the co-simulation that interests us. The step concerning the creation of a variable load will therefore be explained in the following SECTION 3.3.

Note that when running the python code, the entire network is stored in a tabular dataframe and then the power flow can be applied on that dataframe. Actually, a pandapower network is a tabular data structure, based on the pandas library, in which each type of element is an element table that consists of a column for each parameter and a row for each element.

3.3 Integration within mosaik

After testing the pandapower package on its own and becoming familiar with its use, the next step was to create the co-simulation with mosaik as the core element.
The current section is dedicated to the integration into mosaik of pandapower, a pseudo simulator for distribution and a results storage package. The integration of control and communication is described in SECTIONS 3.4 and 3.5 respectively.

3.3.1 How simulators are added within mosaik

To be able to use simulation tools in a co-simulation whose framework is mosaik it is necessary to make these tools accessible and controllable by mosaik, *i.e.* to integrate the simulation models into the mosaik ecosystem. This can be done by implementing the mosaik API for each of

these tools. This API will define the communication protocol between mosaik and its coupled simulation tools. There are two types of API, the low-level API and the high-level API, both of which are shown in FIGURE 3.6 from the mosaik documentation [9].

To summarize, the low-level API exchanges messages coded in JSON using ordinary network sockets while the high-level API consists of an implementation of the former in a particular programming language (at the moment it is only available for Java and Python). The high-level API encapsulates the networking parts and comes with an abstract base class with some methods that need to be implemented in the potential subclasses.

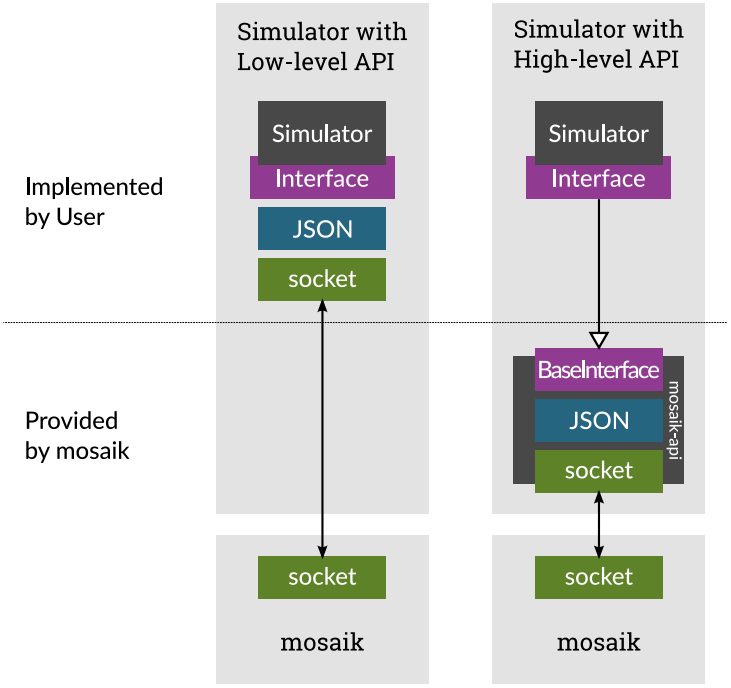


FIGURE 3.6: Differences between the low-level and high-level mosaik API. [9]

In the context of my Master's thesis, it was decided to use tools that were already part of the mosaik eco-system. Nevertheless, note that very slight modifications were made to some of the code that did not work as expected. These changes are notified in the following sections according to their appearance in the use case design's steps.

The different actors of the co-simulation, other than mosaik and before adding control and communication, are therefore the following:

- pandapower for modelling the electrical system and performing power flow computations. Its integration was done thanks to the *mosaik-pandapower* package containing the adapter to connect pandapower to mosaik [152];
- the pseudo simulator for residual load that serves mosaik with load profiles and is therefore used to model the distribution system attached to the `dist_bus` by a time-varying load. This pseudo simulator is encapsulated in the *mosaik-householdsim* package [153];
- and finally, the *mosaik-hdf5* package was used to store the simulation data in an HDF5

database[1] [155].

3.3.2 Co-simulation design

With mosaik, the first step in designing a co-simulation is to configure the different simulators. For the considered use case, this was done in the following way:

```
SIM_CONFIG = {
    'PandaPower': {
        'python': 'mosaik_pandapower.simulator:Pandapower'
    },
    'HDF5': {
        'cmd': 'mosaik-hdf5 %(addr)s'
    },
    'HouseholdSim': {
        'python': 'householdsim.mosaik:HouseholdSim',
    },
}
```

This involves specifying to mosaik which simulators are available and how to start those simulators. In the case where the simulator is, like mosaik, written in Python 3, the co-simulation orchestrator simply imports it and runs it in-process. For example, for pandapower, mosaik imports the `simulator.py` script from the `mosaik_pandapower` module and instantiates the Pandapower class from that script. For householdsim the principle is identical while the storage package `mosaik-hdf5` is launched as an external process communicating with mosaik through sockets.

The next step is to instantiate a `World` object which will contain all the states of the simulation and offer most of the functionality of the co-simulation. It is this central object that is notified of which simulators are started, which entities have been created and the various connections between these entities. In the use case this was simply done this way:

```
world = mosaik.World(SIM_CONFIG)
```

Once the `World` object has been created, the scenario design can begin. First of all, the simulators must be started with the `start()` function via which the name given during the configuration and possible parameters of the simulators can be specified:

```
# Start the simulators.
pandapower = world.start('PandaPower', step_size=15*60, mode='pf')
householdsim = world.start('HouseholdSim')
db = world.start('HDF5', step_size=15*60, duration=END_SIM)
```

In my scenario, it was first decided to run a power flow (as can be seen with the `'pf'` mode) and save the results (HDF5) every 15 minutes, which corresponds to the load variations given by householdsim. Note that pandapower provides the possibility to run time series power flows but since it had been decided to model the load variation via another (pseudo) simulator, one

[1] HDF5, which stands for Hierarchical Data Format 5, is a file format for saving and structuring files containing a very large amount of data. An HDF5 file is therefore a file container [154]

needed to use a simple power flow launched every step size. It can be seen that a variable `END_SIM` is provided to the data storage "simulator" and this variable, which is specified on top of the file, gives (in seconds) the duration of the simulation.
By starting the simulators, objects called `ModelFactory` are created and these enable to instantiate model entities:

```
# Instantiate model entities.
grid = pandapower.Grid(gridfile=GRID_FILE).children
houses = householdsim.ResidentialLoads(sim_start=START,
                                       profile_file=PROFILE_FILE,
                                       grid_name=GRID_NAME).children
db = hdf5.Database(filename=result_file)
```

Here it can be observed that the pandapower, householdsim and hdf5 simulator entities have been created using model parameters. Among these parameters there are input files whose content will be parsed in `create()` functions of *mosaik_API*-suitable simulators. Then this parsed content is used to initialize the simulators in question. The content of the respective files is further discussed below.

First, the parameter passed to the hdf5 entity is simply the name of the file in which the results must be stored.

Then for pandapower, the parameter corresponds to the input file containing all the characteristics of the power system to be modelled.
The *mosaik-pandapower* package effectively proposes to provide the network topology as an input file, either in JSON format, or an Excel file, or by specifying one of the many standard grids existing in the pandapower library. When creating the `grid` entity, the input file, regardless of its format, will be converted into a grid in pandapower format, *i.e.*, as explained in SECTION 3.2.2, into a tabular pandas data structure.
For this input file creation step, several possibilities were therefore presented: either use one of the existing grids, or write/fill a JSON or Excel file. Since a particular use case was decided, the first option was quickly forgotten and a JSON or Excel file had to be created.

Creating such a file from scratch can be very slow, especially as for JSON you need to know the format and how to structure the file. Nevertheless, pandas provides the `to_json()` and `to_excel()` functions and the idea was to use the file created when getting started with pandapower (see SECTION 3.2.2) and use `to_json()` to convert the resulting dataframe into a JSON file.
With this technique, I went from a code generating a dataframe, to a JSON file and then from this file to the same dataframe. It is not the smartest way to proceed but since the first step of the practical part was to get familiar with pandapower and to make the use case within pandapower, the code was already ready to use and it turned out to be the simplest method for this case.

Other not too complicated possibilities would have been to create the JSON from an empty network (still via a Python script but only by calling the `pandapower.create_empty_network()` function) and to manually fill in all the fields that would already exist. For Excel this option is not feasible because creating an empty network and passing it to `to_excel` does not create the fields for all possible elements. However, this format is much more easily readable and

manipulable by the user, which makes it an interesting option. A final and simplest choice would obviously have been to use a standard grid from the pandapower library.
Note that the pandapower documentation [12] compares Excel and JSON formats for saving and loading grids. Excel is said to be more human readable but induces long load and save times and requires libraries that are not part of the standard Python distribution. For JSON files, an advantage is that they can be interpreted in other languages but a disadvantage is the possible insecurity with additional translation in JSON notation.

A final important note to be made in relation to the input file that was supplied to mosaik once generated by the pandapower code is that, prior to performing this operation, the distribution bus load was set to 0 MW. Indeed, this was done as it had been decided to simulate the distribution via an additional (pseudo) simulator. However, the load was not removed, only set to 0 MW because householdsim will be connected to it.

Finally, concerning the parameters passed to householdsim, the main one corresponds to the file containing the load profiles. The other two will be used to parse the data from this profile file and are the simulation start date (in the form "*yyyy-mm-dd hh:mm:ss*") and the name of the power system being modelled by pandapower.

The file should contain a set of load profiles for a given period, a list of IDs describing which profile is attached to which power system node, some meta data and attributes such as the number of separate households attached to the node and the number of residents per household.
Here, the file was created based on the file used in the mosaik demonstration [156], by modifying the list of IDs with the name of the power system under consideration and the name of the node to which the load is attached. The structure of the first few lines of the file used can be seen in FIGURE 3.7 below.

```
# meta
{"start_date": "2014-01-01 00:00", "resolution": 15, "unit": "MW", "num_profiles": 60}
# id_list
{
"grid_file": ["dist_bus_load"]
}
# attrs
num_residents   ,       2,       2,       2,       2,       2,       4,       2,       4,
num_hh          ,       1,       1,       1,       2,       1,       1,       1,       2,
# profiles
2014-01-01 00:00,  136.76,   41.34,  152.66,  261.84,   62.40,  188.53,  177.52,  558.59,
2014-01-01 00:15,  330.67,  319.30,  249.48,  432.85,   62.64,  266.66,  403.14,  494.19,
2014-01-01 00:30,  230.0,  345.85,  260.34,  486.18,  153.67,  223.28,  335.08,  564.88,
2014-01-01 00:45,  50.72,  442.71,  165.64,  553.54,  153.14,  227.09,  515.32,  793.35,
2014-01-01 01:00,  340.48,  322.18,  164.73,  438.62,  449.57,  270.88,  396.80,  682.30,
2014-01-01 01:15,  137.80,  115.81,  228.50,  262.88,  509.75,  295.65,  178.56,  853.45,
2014-01-01 01:30,   62.75,  116.02,  229.10,  260.23,  277.32,  268.06,  178.13,  466.77,
2014-01-01 01:45,   62.41,  117.21,  261.72,  225.51,  153.38,  185.72,   89.51,  338.48,
2014-01-01 02:00,   63.91,  192.82,  182.62,  552.28,  152.55,  147.67,  199.78,  593.05,
2014-01-01 02:15,  328.14,  191.96,   87.24,  331.37,   62.62,  184.91,  198.54,  450.99,
2014-01-01 02:30,  330.53,  169.87,  152.71,  261.31,   63.53,  190.03,  177.90,  540.69,
2014-01-01 02:45,   40.47,   38.03,  153.77,  261.38,   63.54,  267.69,  255.04,  526.25,
2014-01-01 03:00,  139.53,   39.57,   88.97,  224.86,  152.12,  223.60,  166.68,  393.93,
```

FIGURE 3.7: First lines of the profile input file for householdsim.

This file is in data or gz.data format and in the list of IDs, the first entry (unique in the current use case) refers to the first entry in the list of profiles, the second entry to the second

load profile and so on. In our case, having reused the file from the demonstration, there were far too many profiles (characterised by the "num_profiles" field) for the single bus considered but this is not a concern because by providing x nodes in the ID list, householdsim only takes into account the first x profiles, *i.e.* the first x columns.

Then, after having created entities for the grid, the residential consumption (`houses`) and the database, these entities had to be connected to each other. First, the entity representing residential consumption is connected to `dist_bus`:

```
buses = filter(lambda e: e.type == 'Load', grid)
buses = {b.eid.split('-')[1]: b for b in buses}
house_data = world.get_data(houses, 'node_id')
for house in houses:
    node_id = house_data[house]['node_id']
    world.connect(house, buses[node_id], ('P_out', 'p_mw'))
```

This connection was made by manipulating the `grid` and `houses` entities to obtain the scenario entity corresponding to the distribution bus and the one corresponding to the load profile. Then the output attribute P_{out} of the source `house` scenario entity has been connected to the input attribute p_{mw} of the destination entity `dist_bus_load`, meaning that during the co-simulation execution, pandapower uses the P_{out} values of householdsim as inputs for its p_{mw} attribute of `dist_bus_load` element.

Secondly, the results of the power flow calculations performed within pandapower were to be connected to the hdf5 database. In this case, it was chosen to save the state of the different buses, i.e. the active and reactive power consumed (negative result when the bus in question produces power) as well as the voltage amplitude and angle at these buses at the end of the power flow calculation:

```
nodes = [e for e in grid if e.type in 'Bus']
connect_many_to_one(world, nodes, hdf5, 'p_mw', 'q_mvar', 'vm_pu', 'va_degree')
```

Finally, once all the simulators had been started, the simulation entities instantiated and connected to each other, the simulation could be started for a duration of "`END_SIM`" thansk to the following simple line:

```
world.run(until=END_SIM)
```

3.4 Adding control

After successfully completing the co-simulation combining pandapower, householdsim and data storage within mosaik, the next step was to include the control to that combination. As a reminder, the goal was to control the tap position of the transformer based on voltage measurements taken at a measurement bus located between the secondary of the transformer and the bus connected to the residential consumption.
Note that the current section does not include the add of communication layer to the system since this step was carried out after ensuring that the control was functional and gave the desired results.

In addition, it is important to note that the first version of the controller that was implemented was reworked and modified when it came to taking communication into account. For the sake of clarity and in order not to confuse the reader, the version without communication is not described in detail here but the most important information is given to understand the thought process. On the other hand, the "calculation" part of the control mechanism has not been altered from one version to the other and is therefore detailed in the current section.

3.4.1 Control implementation

This section thus describes the implementation of the controller described here-above. This controller is provided by a python code adapted to the mosaik Sim API, importing the package `mosaik_api` and whose class `LoadTapController` has to implement in particular the methods `create()` and `step()` of the class `mosaik_api.Simulator`.

In this first version, unlike what will follow, the idea was that two control agents would be created when designing the scenario in mosaik. These agents were two as the first was to take care of the voltage reading at the measurement bus and the second was to read the tap position (and the minimum and maximum possible tap positions) at the transformer and then send back the new tap position after "concerting" the data with the measurement bus agent. The principle was therefore to be able to instantiate agents whose parameter called *monitored* indicated whether it was the agent in charge of the transformer or the measurement bus, in order to be able to manage the correct reading of inputs in the controller's `step()` method.

However, after observing that the first results did not follow the logic set up, it was realised that when running the co-simulation, the value of `tap_pos` sent by pandapower to the controller always remained the same. And this even after it had been changed during the previous controller run and successfully sent to pandapower. In fact, the value of the `tap_pos` variable sent by the controller was correctly added in the `net` dataframe of pandapower and taken into account in the power flow. The problem was elsewhere: the `tap_pos` variable is a parameter of the "*static*" field of the transformer and when connecting to mosaik, it is the initial value that is sent. This is different for the voltage value which is actually a "result" variable whose value is changed after a power flow calculation.

This is why the values of `tap_pos`, `tap_min` and `tap_max` (the last 2 being anyway considered as static from the start) sent by pandapower to the controller were only taken into account during the first call to `step()` and were directly placed in instance variables of the same names. The instance variable `tap_pos` is therefore the one on which the modifications are made and whose potential new value is sent back to the transformer. Note that it was therefore possible to remove the idea of the 2 agents and keep only one, but this idea was kept for the second version of the `LoadTapController` described below.

Then within the `step()` method, *i.e.* the method called by the mosaik scheduler during runtime, after having read the input values and placed them in the appropriate variables, the actual control was created. This is as follows:

```
vm_acceptable = self.check_voltage(vm_pu, self.tap_pos, self.tap_max, self.tap_min)
if not vm_acceptable:
    self.tap_pos = self.control_step(vm_pu, self.tap_pos, self.tap_max, self.tap_min)
```

```
else:
    yield self.mosaik.set_data(commands)
```

where `vm_pu` is the voltage (in per unit) at the measured bus and the methods `check_voltage` and `control_step` were inspired from the `DiscreteTapControl` class of pandapower ([157]) and are given by:

```
    def check_voltage(self, vm_pu, tap_pos, tap_max, tap_min):
        """
        Checks if the voltage is within the desired voltage band
        """
        if vm_pu < self.vm_lower_pu and tap_pos == tap_min:
            print("Cannot go beyond the minimum tap position of the transformer")
            return True
        elif vm_pu > self.vm_upper_pu and tap_pos == tap_max:
            print("Cannot go beyond the maximum tap position of the transformer")
            return True
        return self.vm_lower_pu < vm_pu < self.vm_upper_pu

    def control_step(self, vm_pu, tap_pos, tap_max, tap_min):
        """
        Implements one step of the OLTC, one tap position up or down
        """
        if vm_pu < self.vm_lower_pu and tap_pos > tap_min:
            tap_pos -= 1
        elif vm_pu > self.vm_upper_pu and tap_pos < tap_max:
            tap_pos += 1
        return tap_pos
```

It can be seen that if the voltage was acceptable, a (empty) `commands` dictionary was sent to mosaik (thanks to `set_data`) which will in turn place it as input to the power system simulator during its next step. In case the voltage exceeded the upper limit or was below the lower limit, the new tap position is entered in `commands` with the IDs of the agent monitoring the transformer and the model to which it is attached (*i.e.* the pandapower's transformer entity). And after that, the command is sent to mosaik.

```
commands[agent_eid][model_eid]['tap_pos'] = self.tap_pos
```

One important thing to note is that in addition to the design of the controller, a small modification had to be made to the *mosaik-pandapower* package. When the controller was first tested, the results were inconclusive and the voltage was not corrected. After searching for the problem, it turned out that in the `set_inputs()` method of the `mosaik_pandapower.model.pandapower` class, adding a new tap position was not convenient and did not work. Therefore two lines of code were added, the last two lines of this piece of code:

```
    def set_inputs(self, etype, idx, data, static):
        """setting the input from other simulators"""
        name = list(data.keys())[0]
        value = list(data.values())[0]
        "..."
```

```
            "..."
            elif etype == 'Transformer':
                if name == 'tap_pos':
                    self.net.trafo.at[idx, 'tap_pos'] = int(value)
```

so that it worked with the on load tap changer controller that was built.

3.4.2 Control mechanism integration to the co-simulation

The integration of the `LoadTapController` into the scenario proceeded in the same way as for the other co-simulation participants. First mosaik was told how to start it, then it was started and had to be initialised with a step size value for the simulation as well as with values of voltage magnitude lower and upper limits that are used to control the measured voltage.

Concerning the step size, the same value as for the pandapower step size was used as the control requires power flow results and as running the controller several times between two power flow calculations is not useful. In addition, this step size value is lower than in the case without control. Indeed, it was necessary that the check took place more than once per load value (*i.e.* more than once per 15min) in order to be able to verify that the new calculated tap position was sufficient (since the tap can only be changed once per check step) to bring the voltage back within the accepted limits before the load changes again. The idea to find the step size value was to divide 15 minutes by the maximum possible number of tap changes. This number is the difference between the maximum tap position and the minimum tap position, $tap_{max} - tap_{min} = 18$. This divisor was however set equal to 20 to obtain an integer step size equal to $\frac{15 \cdot 60}{20} = 45$ seconds. In this way, the controller is sure to do its job while not running it and pandapower an infinite number of times. This is still very often (if not always) too many times, but in the small use case that is considered here this is not a problem.

`agent_tfo` and `agent_bus` entities were then instantiated and connected to the transformer and measurement bus pandapower entities respectively as shown below:

```
# Transformer
tfo = filter(lambda e: e.type == 'Transformer', grid)
tfo = {t.eid.split('-')[1]: t for t in tfo}
world.connect(tfo['tfo_hv_mv'], agent_tfo, ('tap_pos','tap_pos'),
                ('tap_max','tap_max'), ('tap_min', 'tap_min'),
                async_requests=True)
# Bus
meas = filter(lambda e: e.eid == '0-meas_bus', grid)
meas = {m.eid.split('-')[1]: m for m in meas}
world.connect(meas['meas_bus'], agent_bus, ('vm_pu', 'vm_pu'),
                async_requests=True)
```

One difference from the connections established in the first scenario is that the `async_requests` flag has been set to "True". In mosaik it is forbidden to create cyclic data-flows by connecting an entity A to an entity B and then entity B to A because the scheduler does not know which calculation to perform first. One way to solve this problem (the other one being introduced in Section 3.5) is to use the `async_requests` flag which will allow to use asynchronous requests: the first simulator of the connection will send its inputs in the usual way and then when

the second one is executed, the call to `mosaik.set_data()` will send the result of the second simulator to the first one before continuing its execution.

3.5 Adding communication

The final step in building the cyber-physical co-simulation use case that was decided upon was to add the cyber layer to the combination that had just been verified to work.
As noted at the beginning of this chapter, the communication layer was modelled and simulated using the existing *mosaik-communication* suite [158]. Recall that this basic communication suite is far from being of the same level as simulators such as NS-3. But as already mentioned, the main part of this Master's thesis was focused on research and this suite was used in order to add some communication despite the limited time dedicated to the practical part. Note that this suite has been developed recently and is still under development, and a certain number of modifications have been applied, which are explained in this section.

3.5.1 `LoadTapController` modification

As written in the previous section, the code of the `LoadTapController` has been modified somewhat for this final version. The control mechanism is still exactly the same but the idea of using an agent for the transformer as well as an agent for the measurement bus has been dropped in favour of a single control agent. Therefore the differentiation parameter "*monitored*" has been removed as well as the attributes `tap_max` and `tap_min`. These, along with the initial position of the transformer tap, are passed to the control simulator during its initialisation, after being loaded from the grid file into the mosaik scenario code.
The difference this makes is that the `step()` method should only receive as input the `vm_pu` attribute of the measured bus voltage. For the rest of the `LoadTapController` class, things are unchanged except for a slight adaptation concerning the temporal aspect, but this is explained in the following Section 3.5.4.

3.5.2 Introduction to *mosaik-communication* and its adaptation

In this section, the most important points regarding the communication suite, more precisely regarding the modifications made to obtain the
`mosaik_communication.comm_simulator_modified.CommSimulator` class, are presented but the functioning is not described in detail. As for the temporal management and the adaptations made to make the `CommSimulator` class synchronised in the use case co-simulation, it is developed in the dedicated Section 3.5.4.

To begin with, the function `init()` has been adapted with the addition of the `step_size` argument which will be useful further. In this function, it was also a question of slightly modifying the way of creating the dictionary instance variables containing information on the connections existing in the communication network.
To build the dictionary instance variables in question, the communication information is passed to the simulator via a communication network configuration file. This file is in JSON format and contains information about the communication nodes to be created, the connections between these nodes and the attributes that can be communicated via these connections. The content of the file used is shown in Figure 3.8 below, where it can be seen that a delay value has been added to the connections. This value is optional and can potentially be accompanied by

parameters altering the information transmitted, *i.e.* "*factor*" and "*offset*", serving respectively to multiply the value of the communicated attribute and to add an offset to it.

```
{
"nodes": [
    "node_meas_bus",
    "node_ctrl",
    "node_tfo"
    ],
"connections": [{"node_meas_bus->node_ctrl": 2}, {"node_ctrl->node_tfo": 2}],
"attributes": [
    "vm_pu",
    "tap_pos"
]
}
```

FIGURE 3.8: Communication network configuration file.

Note that care must be taken when creating the JSON file as format errors often appear when parsing via the JSON decoder depending on added spaces, line breaks, etc.

The other modifications made to the `CommSimulator` class took place in the `step()` function which is launched during the execution of the co-simulation. These modifications concern the management of time with the addition of a flag in the form of an instance variable as well as certain conditions during the processing of inputs. These are explained in more detail in SECTION 3.5.4 below.

Concerning the way of integrating the simulator in the mosaik scenario, it was done in the same way as for the others except that the configuration file of the communication network was passed when the simulator was started. Then, a scenario entity `comm_entity` was instantiated from the `ModelFactory` object created by the simulator start.

Note that the version of the code *comm_simulator_modified.py* has been adapted by focusing on the use case in which a control is applied and is therefore quite specific to the control case under consideration. It therefore lacks genericity and could be made more generic in future development but here the main purpose was to test the feasibility of co-simulation around mosaik.

3.5.3 Entities connection and co-simulation structure

The connections that have been added and/or modified with the addition of the communication simulator are described in this subsection based on the system diagram that was shown previously and was adapted with simulator names in FIGURE 3.9.

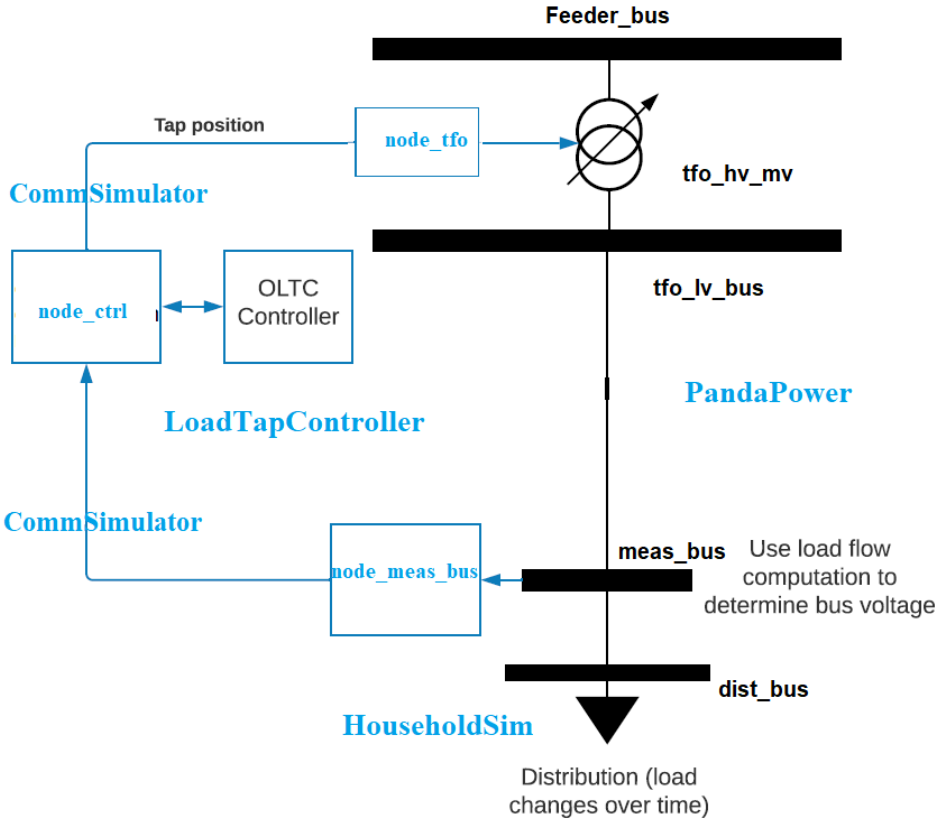


FIGURE 3.9: Cyber-physical use case topology.

As a reminder, the scenario entities available before establishing are the following:

- a list of entities `grid` containing all the entities of the power system elements;
- a list of entities `houses` containing a single entity modelling a variable load;
- an `agent` entity for control;
- a `comm_entity` entity representing the communication network;
- and finally a `db` entity for the storage database.

To begin with, the connection between the single householdsim residential consumption pseudo-simulator entity and the `grid` distribution bus as well as the connection between the power `grid` buses and the `db` HDF5 storage have not changed. What was changed in the rest of the scenario is due to the appearance of the "`comm_entity`" containing communication nodes as well as the disappearance of one of the two agents compared to the co-simulation without communication described earlier in the paper.

As observed in FIGURE 3.9, the electrical system had to be connected to the communication system in order to send information to the intelligent electronic device (IED) modelled by the `agent`.

The connection of the power grid to the communication network was first established in the following way:

```
comm_nodes = {e.eid: e for e in comm_entity.children if e.type == 'node'}
# Bus
meas = filter(lambda e: e.eid == '0-meas_bus', grid)
meas = {m.eid.split('-')[1]: m for m in meas}
world.connect(meas['meas_bus'], comm_nodes['node_meas_bus'], ('vm_pu', 'vm_pu'))
# Transformer
tfo = filter(lambda e: e.type == 'Transformer', grid)
tfo = {t.eid.split('-')[1]: t for t in tfo}
world.connect(comm_nodes['node_tfo'], tfo['tfo_hv_mv'], ('tap_pos','tap_pos'),
              time_shifted=True, initial_data={'tap_pos': tap_pos_init})
```

where `comm_nodes` is a dictionary containing information about the different network nodes introduced by the configuration file of FIGURE 3.8. It is observed that the electrical measurement bus has been connected to the `node_meas_bus` node of the communication network with the attribute `vm_pu` sent from one to the other. Then, the communication node `node_tfo` was connected to the transformer to be able to transmit the position of the tap. The particularity of this connection concerning the presence of the flag `time_shifted` and the parameter `initial_data` is explained in the section dedicated to the temporal aspect of this co-simulation.

As for the connection between the communication network and the controller, it was established in the following way:

```
world.connect(comm_nodes['node_ctrl'], agent, ('vm_pu', 'vm_pu'),
                  async_requests=True)
```

where this time the node `node_ctrl` has been connected to the agent with the particularity of allowing asynchronous requests, which allows the specified attribute, *i.e.* `vm_pu`, of the node to be sent to the agent but also to return an attribute to that agent (and the design of the `LoadTapController` has been made so that this returned attribute is the position of the tap).

To conclude, the connections made were therefore intended so that, during the execution of the co-simulation, the voltage of the measurement bus obtained after a power flow calculation was collected by `node_meas_bus`, which transmitted the information to `node_ctrl` thanks to the `node_meas_bus->node_ctrl` connection of the communication network. Then the information had to be sent from `node_ctrl` to `agent` before being processed and sent back (this time with the tap position) to the same node. Finally, the information had to pass through the communication link `node_ctrl->node_tfo` before being provided to the transformer through the connection between it and `node_tfo`.

To make the link with the diagram in FIGURE 3.9 representing the whole cyber-physical system, it can be seen that the Merging Unit is modelled by a node, just like the other points of the communication network. More realistic device modelling taking into account protocols and other specifics of the communication network is possible with reference simulators such as NS-3, but in this case a simplified communication taking into account only delays has been simulated. For a more realistic modelling and simulation taking into account these specificities, research has been conducted to help the future of the CYPRESS project and the results of this research are expressed in the final Chapter 4.

3.5.4 Time handling within the co-simulation

A final step in the design of the co-simulation scenario was the management of the timing aspect. Although mosaik takes care of the synchronisation and data exchange, it still needs to be provided with the appropriate information to make this possible. This information takes the form of passing different step size values to the simulators and potentially adding conditions and exceptions to their APIs.

As explained in SECTION 2.4.4, the time course of a co-simulation in mosaik is done with a discrete event-based approach in which the frequency of calls to the different simulators can vary during the runtime. This frequency will depend on the step size provided at initialization and the operations performed when calling their `step()` method. Let us take the general example of the temporal evolution of a simulator whose `step()` method looks like this:

```
def step(self, time, inputs):
    """""""""""""""""""
    |...operations...|
    """""""""""""""""""

    if "...":
        "..."
    else:
        "..."
        return time

    """""""""""""""""""
    |...operations...|
    """""""""""""""""""

    return time + self.step_size
```

When this method is executed, its current simulation time is provided and following the operations required to perform the current step, the time at which the next step should be taken is returned by the method. This example is illustrated in TABLE 3.3 below.

Call to `step()`	Current time	Next time (what is returned)
1st	0	`return time + step_size`
2nd	`step_size`	`return time + step_size`
3rd	2 `step_size`	`return time`
4th	2 `step_size`	`return time + a`
5th	2 `step_size` + `a`	...

TABLE 3.3: Example of the temporal aspect of a simulator.

To manage data exchanges between different simulators, mosaik is based on the fact that a simulator can run only if all input data are available. This is why the chosen step sizes, implemented operations and the available mechanisms provided by mosaik had to be carefully manipulated. The time management of the different participants in the co-simulation is given below simulator by simulator.

HouseholdSim

For the load profile simulator connected to the electrical distribution bus, the frequency of its calls depends on the time resolution entered in the metadata of the profile file, as seen in FIGURE 3.7. In the current use case, a resolution of 15 minutes was entered, so that a new load value is output every 15*60 seconds.

```
def step(self, time, inputs=None):
    """""""""""""""""""
    |...operations...|
    """""""""""""""""""
    return time + (self.model.resolution)*60
```

PandaPower

The power flow simulator, on the other hand, is simply called every `step_size` seconds and this value is not altered by the operations performed within it.

```
def step(self, time, inputs):
    """""""""""""""""""
    |...operations...|
    """""""""""""""""""
    return time + self.step_size
```

However, it was necessary to take into account and analyse all the connections established between the grid and the other simulators. Indeed, this simulator is connected to:

- HouseholdSim to **receive** a load value from the `dist_bus`;
- CommSimulator to **send** the voltage of the `meas_bus` to the `node_meas_bus` **and** to **receive** by `node_tfo` the transformer tap position;
- and finally to HDF5 to **send** the power flow results.

The connections to HouseholdSim and HDF5 are not a problem as they are one-way. However, in order to be able to both send and receive values to CommSimulator without creating problems, the connection had to be adapted. This time this was not done via the `async_requests` flag but rather, as shown in SECTION 3.5.3 and below, via the `time_shifted` flag and the `inital_data` parameter. Thanks to these, mosaik knows that the output (*i.e.* the tap position) of `node_tfo` is to be used for the next time step(s) of pandapower. Moreover, for the first time step, the tap position used is the one provided in `inital_data` since this data cannot yet be provided by the CommSimulator which itself is waiting for the measurement bus voltage value from pandapower.

```
world.connect(meas['meas_bus'], comm_nodes['node_meas_bus'], ('vm_pu', 'vm_pu'))
"..."
world.connect(comm_nodes['node_tfo'], tfo['tfo_hv_mv'], ('tap_pos','tap_pos'),
              time_shifted=True, initial_data={'tap_pos': tap_pos_init})
```

CommSimulator

The communication simulator is the one to which I had to make the most adaptations to manage time because it is the most complex mechanism and the most often called simulator. To begin with, the step size provided at the time of initialisation is the same as for pandapower. On the other hand, exceptions had to be added to the `step()` method, in particular with the addition of a flag informing whether the control (handled by the `agent`) has been carried out or not. These exceptions are not described in detail here because they are rather specific and elaborate and do not really help to understand how co-simulation works. Nevertheless, TABLE 3.4 gives a clearer picture of the desired (and achieved) temporal behaviour of this communication simulator. Note that instead of using the generic "delay link 1", "delay link 2" notations, the table was built with the values assigned in the use case (*i.e.* 2 for both links) for better readability.

time	**`node_meas_bus`**	**`node_ctrl`**	**`node_tfo`**
0	vm_{pu} received from the grid bus	-	initial tap_{pos} sent to the transformer (specified in the time shifted connection)
2	-	vm_{pu} received through the link + vm_{pu} sent to the control agent	-
4	-	tap_{pos} received from the control agent (asynchronous request)	-
6	-	-	tap_{pos} received through the link
45	**new** vm_{pu} reveived from the grid bus (resulting from tap_{pos} change)	-	tap_{pos} sent to the transformer
47	-	**new** vm_{pu} received through the link + **new** vm_{pu} sent to the control agent	-
49	-	**No new tap_{pos} since new vm_{pu} is acceptable**	-

TABLE 3.4: Communication simulator temporal execution when the voltage has to be corrected.

LoadTapController

The step size provided at initialisation of the load tap controller is the same as for pandapower. However, there is a slight difference with the case without communication, which is that the agent is called to run not directly after pandapower but rather after a certain communication delay which appears in the `node_meas_bus->node_ctrl` link. Therefore, an exception was added for `time = 0` to the `step()` method of the `LoadTapController` so that all other

calls to the agent would be time delayed by `delay` seconds with respect to the start of the `node_meas_bus->node_ctrl` communication (and the power flow by extension).

```
def step(self, time, inputs):
    """""""""""""""""""
    |...operations...|
    """""""""""""""""""
    if time == 0:
        return time + 2
    return time + self.step_size
```

HDF5

Finally, as for the "simulator" used to store the results of the co-simulation, it is called at the end of each pandapower round because its step size was initialised as being equal to that of pandapower.

```
def step(self, time, inputs):
    """""""""""""""""""
    |...operations...|
    """""""""""""""""""
    return time + self.step_size
```

In conclusion, synchronisation depends mainly on the step size that has been decided between 2 power flow calculations, *i.e.* the one supplied first to pandapower and then to the controller, communication and storage. In the current implementation, this step size has been chosen to be equal to 45 seconds, the reason for this choice has been given in Section 3.4.2.

3.6 Scenario test and results

Finally, after successfully integrating the communication simulator into the co-simulation already combining the power system, variable load and data storage, the design was fully completed and the use case co-simulation could be executed.
As a reminder, the use case that was decided upon consists of simulating the operation of a small power system made of an interconnection between a high voltage system and a lower voltage system. The higher voltage system can represent the transmission grid acting as a voltage source/feeder and on the other side of the transformer, lines and measurement bus is a bus connected to a load that varies in time, which can represent the distribution system.

The results obtained at the bus at which the voltage was measured to carry out the test are displayed in the following Table 3.5 for the first hour simulated, with the value of the variable load displayed alongside these results. It should be remembered that the upper and lower limits that the voltage could reach were defined as being equal to 1.1 and 0.99 per unit respectively. Also, note that the lines with bold type are those for which the voltage value is outside these limits.

Power flow iteration	**Voltage measured at `meas_bus` [p.u.]**	**Variable load [MW]**
0	0.99667635	50.76
1	0.99667635	50.76
...	...	...
19	0.99667635	50.76
20	**0.907090302**	**300.67**
21	**0.927653307**	**300.67**
22	**0.948301515**	**300.67**
23	**0.969103815**	**300.67**
24	0.990121844	300.67
25	0.990121844	300.67
...	...	...
39	0.990121844	300.67
40	**0.92035108**	**390.49**
41	**0.948347208**	**390.49**
42	**0.975647864**	**390.49**
43	1.002548587	390.49
44	1.002548587	390.49
...	...	...
59	1.002548587	390.49
60	**1.11289629**	**70.72**
61	1.094312675	70.72
62	1.094312675	70.72
...	...	...
79	1.094312675	70.72

TABLE 3.5: Use case results for normal operation of the cyber-physical co-simulation using on-load tap changer control.

To show that the co-simulation works and therefore that the on-load tap changer control works, the load value was directly and abruptly increased from the first quarter hour to the second quarter hour from 50.76 MW to 300.67 MW consumed. It can be seen that for the neutral position, a low consumption of 50.76 MW results in an acceptable voltage of 0.9967 per unit. On the other hand, once the consumption increases at the twentieth iteration, *i.e.* after 15 minutes, the voltage decreases sharply and goes completely out of the imposed limits. The reaction is therefore the expected one: between each power flow calculation, the control varies the tap position to increase the secondary voltage and therefore the measurement bus voltage. The tap position is decremented by one unit each time until an acceptable voltage of 0.9901 p.u. is reached at the 24^{th} iteration.
Then, as the load increases again, the same mechanism occurs and the voltage is rectified. Once the load becomes low again, with 70.72 MW consumed, the voltage becomes higher than the upper limit of 1.1 p.u. as the tap has reached a position far away from the neutral position. Therefore, the position is rectified again by the controller but in the opposite direction this time and the voltage returns to an acceptable value after one iteration.

Now let us consider a failure in the communication network occurring at time t = 0. This failure consists here simply in cutting the communication link between the merging unit of the measurement bus and the intelligent electronic device containing the agent in charge of the con-

trol. To do this, it is sufficient to remove the connection between `meas_bus` and `node_meas_bus` and to run the co-simulation.

Power flow iteration	**Voltage measured at `meas_bus` [p.u.]**	**Variable load [MW]**
0	0.99667635	50.76
1	0.99667635	50.76
...	...	...
19	0.99667635	50.76
20	**0.907090302**	**300.67**
21	**0.907090302**	**300.67**
...	...	...
38	**0.907090302**	**300.67**
39	**0.907090302**	**300.67**
40	**0.785614716**	**390.49**
41	**0.785614716**	**390.49**
...	...	...
58	**0.785614716**	**390.49**
59	**0.785614716**	**390.49**
60	0.994310401	70.72
61	0.994310401	70.72
62	0.994310401	70.72
...	...	...
79	0.994310401	70.72

TABLE 3.6: Use case results when the merging unit-IED link is unavailable during the operation of the cyber-physical co-simulation using on-load tap changer control.

From the results, the impact that this failure of the communication network has on the operation of the electrical system can clearly be seen. Indeed, the monitoring of the voltage is no longer taking place, so when the load is increased the voltage is never rectified. Compared to the lower limit imposed, it even reaches an extremely low value of 0.786 per unit in the third quarter of an hour.

Such a problem in the operation of a real power system can have various negative consequences. Uncorrected overvoltage may degrade insulating materials or damage sensitive (electronic) equipment, while a too low voltage may disrupt or even interrupt the operation of certain components. For example, loads (such as motors) may trip to protect against undervoltage, power electronics in converters may lock up or induction motors may stall. Furthermore, another undesired effect is that too low voltages lead to higher Joule losses in the grid [11].

Chapter 4

Future work and conclusion

After having introduced the subject in Chapter 1 by presenting the need to study the modelling and simulation of cyber-physical interactions within the power system, the cyber layer of which is increasingly present and introduces more and more risks linked to "cyber" failures. After having carried out in chapter 2 an in-depth state of the art on the possibilities existing and/or that can be developed to meet this need for modelling and simulation of cyber-physical power systems. After having finally, in Chapter 3, designed a simulation structure combining these two domains, *i.e.* the power grid and the ICT network, and having tested a small basic use case with this structure in order to present a proof of concept. After these achievements it is time to consider and state what could be done in a future development of the ideas within the CYPRESS project, what needs to be improved, what could be considered.

This concluding chapter is divided into three sections. The first briefly discusses possible developments to make the developed structure more generic and less "use case specific". Then, the second section presents the research that has been carried out on the elements constituting the ICT layer, being deployed in practice in real world systems, in order to allow a more detailed and realistic modelling of the cyber layer within the co-simulation. Section 4.3 finally concludes this chapter and this Master's thesis.

4.1 Making the co-simulation platform more generic can be done by...

As explained in the introduction to the chapter, this section aims to present possible developments to make the practical implementation developed more generic. To start with, the way to make the codes developed in chapter 2 less use case specific is very briefly discussed. Then, in a more general way, a way to make the co-simulation platform more generic in order to be able to integrate other simulators easily or to change the input file format is proposed as a way to follow.

...making the current implementation less use case specific

The first step for a future development of the co-simulation whose design approach has been described in Chapter 3 would be to make the implemented codes more generic and less specific to the use case considered. Indeed, in some places in the implemented codes, the operations performed are quite specific to the use case. For example, the names of the buses assigned

when creating the power system in pandapower are noted as they are in some places instead of extracting them from a parsing of the input files to have something generic independent of the assigned names. Therefore, a first thing to do in the case of code reuse/inspiration would be to make it possible to create other scenarios with other topologies of the power system, communication network, etc. through some adaptations. In this regard, it is worth recalling that pandapower offers a large library with a set of pre-implemented standard networks of all sizes (see [159]).

...going further towards full genericity

Still with the aim of improving genericity, the ideal (in the context of the CYPRESS project) would be to have a modular co-simulation platform allowing to choose which simulator to use, to switch simulators when changing the type of simulation, to switch the input file format, etc. So the reflection is as follows:

Firstly, we would like to have a platform capable of reading input files and converting these inputs into generic models of power systems or ICT networks. To do this, we would have parsers of different types (CSV, text, etc.) that would have to respect a certain interface in order to generate generic models for each domain, which would be identical (for their respective domain) regardless of the initial file's type.

On the side of the simulators to be integrated one should implement an API for each simulator which should also comply with a certain common interface. The aim is to be fed by the generic models created by the parsers. To this end, given that mosaik allows to add extra methods to the API of simulators that comply *mosaik_API*, one could consider having a function that would have to be implemented by each simulator, into which one could pass these generic models (however, it remains to be seen if this is feasible, the feasibility of this step has not been evaluated and is simply proposed here as a possibility to be studied).

The third point would be to find a way to describe a generic scenario capable of handling any power or ICT simulator. Here again we would have a scenario parser that would create the scenario in the format desired by mosaik. We don't want to have to rewrite all the code in python when we change simulators or connections, but rather simply provide a sort of input file that would be a description of the scenario and after parsing, the code would be responsible for providing mosaik with what it needs.

In the end, we would have roughly the mechanism described in FIGURE below: file parsers receive the configuration files from the power and ICT systems to generate the models. Then these models are fed to the simulators and finally mosaik would drive the simulators based on the provided scenario.

Note that the proposed mechanism is close to what is already happening in mosaik except that at the moment, we go directly from the file parser to the simulators and there is no intermediary creation of a generic model that can be used by any simulator (in its respective domain). The aim is to avoid having to change the whole chain from the parser to the simulator API when a new file format is desired.
For example, in the current situation, if one has chosen to use several power system simulators (allowing different types of simulations) and it is decided to change the input file format, then all the APIs will have to be changed. Whereas with the idea proposed here, one will simply

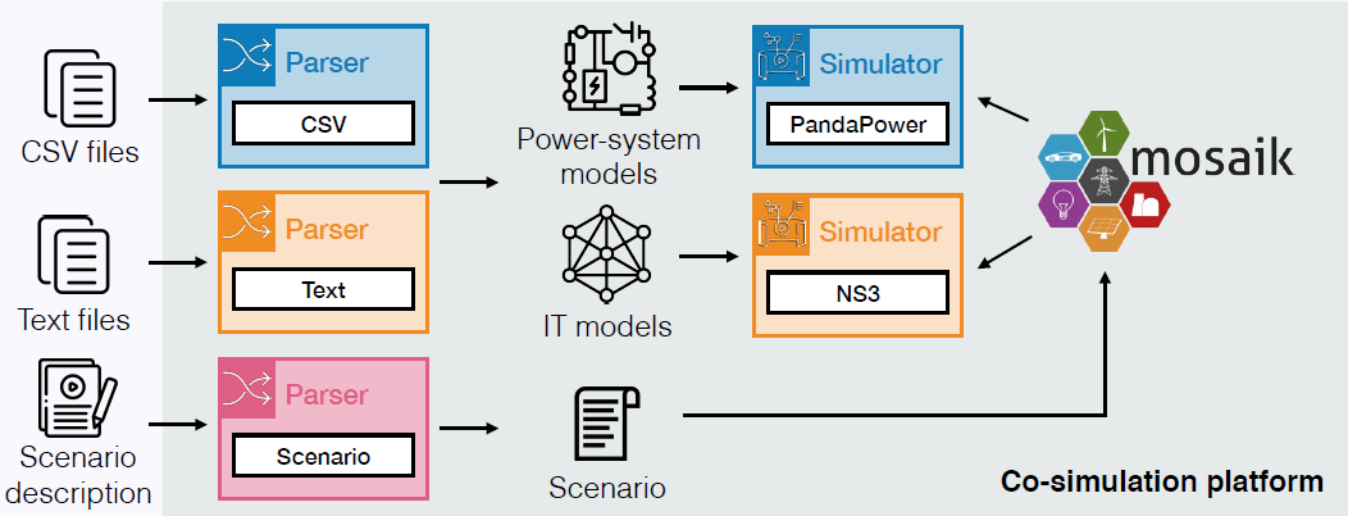


FIGURE 4.1: Possible genericity in the co-simulation platform

have to code the new parser respecting the interface of the parsers, and the generic model will be generated and can be sent to the simulators that already know how to handle these generic models.

4.2 Towards more detailed cyber layer modelling

In the practical case implemented in Chapter 3, the communication layer is modelled in a very simplistic way with only some delay in the information transport links. However, it would be preferable to be able to simulate cyber-physical power system scenarios incorporating the more complex details and characteristics of the ICT layer. Therefore, still with a view to the future development of the co-simulation platform, the idea would be to replace the *mosaik-communication* suite employed by a reference simulator such as NS-3 in order to have the possibility of modelling the ICT system in more detail and to be able to build scenarios closer to real world operation. In this respect, some research has been conducted to investigate some of the ICT elements deployed in practice in cyber-physical power systems and the result of this research is discussed in this section.

Note that NS-3 is mentioned here as a simulator to be integrated in the mosaik co-simulation because firstly it is an advanced, open-source software and widely used in academic research and industry. Moreover, it is often recommended and even considered by many telecom specialists as the best discrete event simulation tool [137]. Secondly, two works on smart grids were found that used NS-3 in a co-simulation whose orchestrator framework is mosaik: [37],[71] and [43],[160]. This means that it can be integrated with mosaik and to achieve this integration, the low-level API discussed in SECTION 3.3 must be implemented.

Another way to make the simulations more realistic is to run scenarios involving simulated cyber-attacks. Indeed, in the test case considered in SECTION 3.6, the communication network failure scenario consisted simply of cutting a link between measurement and control. However, in the real world, as presented in SECTION 1.1, specific cyber-attacks can take place and lead to different consequences.

4.2.1 Elements deployed in real substations

As written above, the addition of a simulator as complete as NS-3 would offer the opportunity to test more complete and realistic communication scenarios. To enable these scenarios to be realised with the co-simulation platform, research work has therefore been carried out, focusing on the search for elements to be modelled for more realistic simulations of communication in cyber-physical power systems. This research focused on substation which is one of the potential sources of cyber-vulnerabilities in cyber-physical power systems.

At the substation level, there is a standard called IEC 61850 which defines information and communication protocols for the automation of electrical substations [19]. This standard is increasingly used as a reference in the automation of electrical substations, but it is also used in other areas such as electric mobility, wind turbines, hydro-generation or distribution automation systems. Indeed, the original (1st edition) IEC 6150 focused on communication at the substation level between relays, control and switchgear to allow interoperability between functions of different vendors. But then the standard was extended to all functions and areas of the network [13].

In an electrical substation there are two types of equipment. Primary equipment such as measurement transformers, switchgear, etc. and secondary equipment such as protection, control and communication equipment. The latter are categorised into three levels in the IEC-61850 standard [150]. These levels are shown are described below.

To start with, in the lowest layer, there is the process level, which contains all switchgear such as circuit breakers, switches and also measuring devices such as current or voltage transformers. Then there is the bay level, which consists of protection, control and monitoring units, *i.e.* intelligent electronic devices[1]. Finally, in the highest layer, there is the station level which contains the equipment for controlling and monitoring the station, *i.e.* the Human Machine Interface (HMI) and the Supervisory Control and Data Acquisition (SCADA) system. These correspond respectively to the interface with the operator at the substation (an operator can control and monitor the substation locally) and the interface between the substation and a remote control centre, allowing the substation automation to be remotely configured and the substation switchgear to be operated [19],[13].

To enable communication between the equipment at the different levels, there are two buses. Firstly the process bus via which the bay level equipment communicates with the process level equipment. Note that to do this, as can be seen in the figure, traditional current and voltage transformers and switchgear can be connected to a Merging Unit (serving as an A/D converter that will provide the appropriate path for messages). Then there is the station bus through which the station level equipment communicates with the bay level equipment.
So in the end, for the highest level (station) to communicate with the lowest level (process), there will first be a message sent to the bay level via the station bus. This will direct the message to the appropriate bay level node which will execute its function and then forward the message to the process level node via, in some cases, a Merging Unit as explained above. Finally, the function will be executed by the process level devices [150].

[1]IED is the new name for relays to reflect their ability to make a decision based on logic and information collected from process level devices.

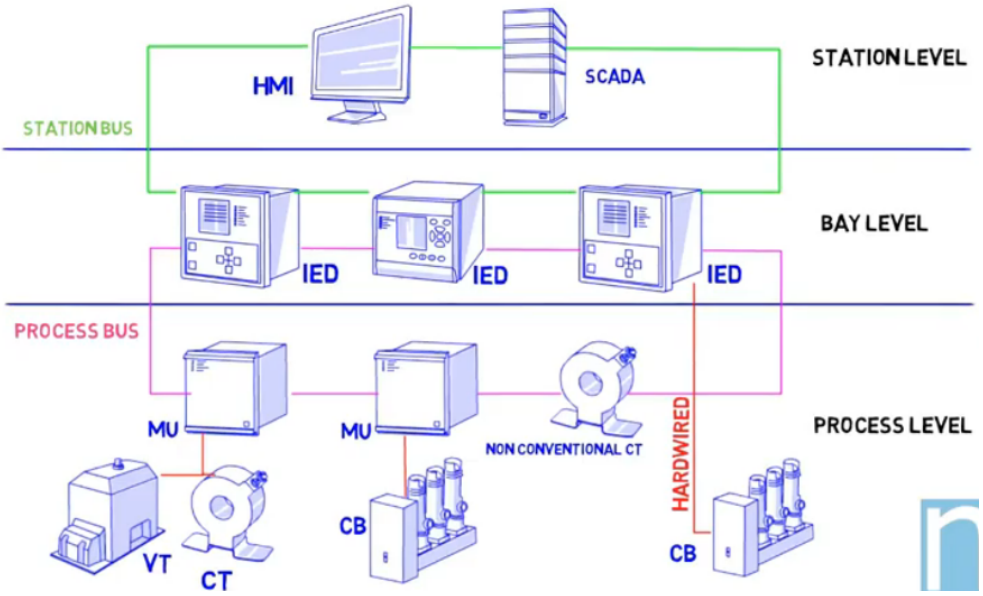


FIGURE 4.2: Substation organisation according to IEC 61850, taken from [13].

4.3 Conclusion

In conclusion of this master thesis, let us recall the objectives that were stated in the introduction and how they were achieved.

As a reminder, the work done in this document was done in order to perform a research task in the CYPRESS fundamental research project. This task, entitled "Co-simulation of cyber-physical transmission", consists, among other things, in selecting a platform adapted to the co-simulation of cyber-physical power systems. My role was therefore mainly to research existing platforms and tools for modelling and simulating these cyber-physical systems.

This has been done through an extensive research of the state of the art of these tools. In this state of the art, I have scanned a wide range of works in order to give an overview of the methods, approaches, tools and frameworks to perform the combined simulation of power and ICT networks. Some choices have been proposed in the different sections, such as using a co-simulated approach given the advantages offered. Other aspects such as specialized frameworks and simulators depend mainly on the desired features of the CYPRESS project and although an attempt has been made to outline a way forward, it is up to the project collaborators to make the most appropriate choices.

For the practical part of my master thesis, some choices have been made in order to implement a use case whose purpose is to serve as a proof of concept of the research and choices made. The implementation was thus built in Chapter 3. The complete process of its realisation has been detailed and the results presented. These results show the feasibility of a co-simulation around the mosaik orchestrator framework. Moreover, these results also show, thanks to a small and simple scenario, the impact that a failure of the communication network can have on the operation of the power grid.

Finally, the question of the genericity of the platform was raised. A proposal to bring this genericity to the current implementation was given in order to give as complete a track as

possible for the future work that will be done in the continuation of the CYPRESS project. To conclude this research work, some elements regarding the more detailed modelling of the cyber layer of the power substations have also been researched and provided as an aid for the simulated scenarios in the future work.

Bibliography

[1] H. Lin, S. S. Veda, S. Shukla, L. Mili, and J. Thorp, "Geco: Global event-driven co-simulation framework for interconnected power system and communication network," *IEEE Transactions on Smart Grid*, vol. 3, pp. 1444–1456, 2012.

[2] K. Mets, J. A. Ojea, and C. Develder, "Combining Power and Communication Network Simulation for Cost-Effective Smart Grid Analysis," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1771–1796, 2014.

[3] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, "EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Transactions on Power Systems*, vol. 21, pp. 548–558, 2006.

[4] V. H. Nguyen, Y. Besanger, Q. T. Tran, and T. L. Nguyen, "On conceptual structuration and coupling methods of co-simulation frameworks in cyber-physical energy system validation," *Energies*, vol. 10, no. 12, 2017.

[5] IEEE Task Force on Interfacing Techniques for Simulation Tools, S. C. Müller, H. Georg, J. J. Nutaro, E. Widl, Y. Deng, P. Palensky, M. U. Awais, M. Chenine, M. Küch, M. Stifter, H. Lin, S. K. Shukla, C. Wietfeld, C. Rehtanz, C. Dufour, X. Wang, V. Dinavahi, M. O. Faruque, W. Meng, S. Liu, A. Monti, M. Ni, A. Davoudi, and A. Mehrizi-Sani, "Interfacing Power System and ICT Simulators: Challenges, State-of-the-Art, and Case Studies," *IEEE Transactions on Smart Grid*, vol. 9, no. 1, pp. 14–24, 2018.

[6] W. Li, A. Monti, M. Luo, and R. A. Dougal, "Vpnet: A co-simulation framework for analyzing communication channel effects on power systems," in *2011 IEEE Electric Ship Technologies Symposium*, pp. 143–149, 2011.

[7] P. Palensky, A. A. Van Der Meer, C. D. Lopez, A. Joseph, and K. Pan, "Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 34–50, 2017.

[8] W. Li, M. Ferdowsi, M. Stevic, A. Monti, and F. Ponci, "Cosimulation for smart grid communications," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2374–2384, 2014.

[9] "Mosaik's documentation - Readthedocs." `https://mosaik.readthedocs.io/en/latest/index.html`, 2020. [Online; accessed 02-June-2021].

[10] L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "pandapower — an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, pp. 6510–6521, Nov 2018.

[11] Thierry Van Cutsem, "Notes du cours ELEC0014 - Introduction to electric power and energy systems," September 2019. Université de Liège - Faculté des Sciences Appliquées - Département d'Electricité, Electronique et Informatique (Institut Montefiore).

[12] "Read the Docs - pandapower - pandapower 2.6.0 documentation." `https://pandapower.readthedocs.io/en/v2.6.0/`. [Online; accessed 11-May-2021].

[13] N2A Consultancy, "10 Min to boost your knowledge on IEC61850."

[14] S. Siraj, A. Gupta, and R. Badgujar, "Network simulation tools survey," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 4, pp. 199–206, 2012.

[15] R. V. Yohanandhan, R. M. Elavarasan, P. Manoharan, and L. Mihet-Popa, "Cyber-physical power system (cpps): A review on modeling, simulation, and analysis with cyber security applications," *IEEE Access*, vol. 8, pp. 151019–151064, 2020.

[16] P. A. Oyewole and D. Jayaweera, "Power system security with cyber-physical power system operation," *IEEE Access*, vol. 8, pp. 179970–179982, 2020.

[17] P. Palensky, E. Widl, and A. Elsheikh, "Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 3, pp. 318–326, 2014.

[18] H. Georg, C. Wietfeld, S. C. Müller, and C. Rehtanz, "A HLA based simulator architecture for co-simulating ICT based power system control and protection systems," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pp. 264–269, 2012.

[19] Thoelen, Klaas and Francart, Jonathan and Karangelos, Efthymios, "Cypress - T1.1 - Introduction to Cyber Systems." Unpublished, April 2021.

[20] D. Rehak, M. Hromada, and T. Lovecek, "Personnel threats in the electric power critical infrastructure sector and their effect on dependent sectors: Overview in the czech republic," *Safety Science*, vol. 127, p. 104698, 2020.

[21] hannover re, "Disruption of critical infrastructure – Power Blackout Risks." `https://www.hannover-re.com/180580/disruption-of-critical-infrastructure-2021.pdf`, 2021.

[22] Congressional Research Service, "Texas Power Outage: Implications for Critical Infrastructure Security and Resilience Policy." `https://crsreports.congress.gov/product/pdf/IN/IN11629`, March 2021.

[23] G. B. Anderson and M. L. Bell, "Lights out: impact of the august 2003 power outage on mortality in new york, ny," *Epidemiology (Cambridge, Mass.)*, vol. 23, no. 2, p. 189, 2012.

[24] Wikipedia contributors, "Northeast blackout of 2003 — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Northeast_blackout_of_2003&oldid=1027417645`, 2021. [Online; accessed 9-June-2021].

[25] Sibelga, "Why does the electricity grid have to stay in balance? - Energuide." https://www.energuide.be/en/questions-answers/why-does-the-electricity-grid-have-to-stay-in-balance/2136/, 2021. [Online; accessed 9-June-2021].

[26] Smart Energy International, "Protecting our critical water utility infrastructure from attack," January.

[27] J. E. Sullivan and D. Kamensky, "How cyber-attacks in ukraine show the vulnerability of the us power grid," *The Electricity Journal*, vol. 30, no. 3, pp. 30–35, 2017.

[28] G. Erbach and J. O'Shea, "Cybersecurity of critical energy infrastructure," 2019.

[29] T. D. Le, A. Anwar, S. W. Loke, R. Beuran, and Y. Tan, "Gridattacksim: A cyber attack simulation framework for smart grids," *Electronics*, vol. 9, no. 8, 2020.

[30] "Elexon, maillon crucial du réseau électrique au Royaume-Uni, victime d'une cyberattaque," *L'Usine Digitale*, 2021.

[31] "Cypress | Smarter electric power systems." https://cypress-project.be/, 2021. [Online; accessed 09-June-2021].

[32] L. Lugaric, S. Krajcar, and Z. Simic, "Smart city — platform for emergent phenomena power system testbed simulator," in *2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, pp. 1–7, 2010.

[33] Hua Lin, S. Sambamoorthy, S. Shukla, J. Thorp, and L. Mili, "Power system and communication network co-simulation for smart grid applications," in *ISGT 2011*, pp. 1–6, 2011.

[34] K. Mets, T. Verschueren, C. Develder, T. L. Vandoorn, and L. Vandevelde, "Integrated simulation of power and communication networks for smart grid applications," in *2011 IEEE 16th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 61–65, 2011.

[35] S. Tan, W. Song, Qifen Dong, and L. Tong, "Score: Smart-grid common open research emulator," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pp. 282–287, 2012.

[36] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pp. 1–7, 2008.

[37] A. G. Wermann, M. C. Bortolozzo, E. Germano da Silva, A. Schaeffer-Filho, L. Paschoal Gaspary, and M. Barcellos, "ASTORIA: A framework for attack simulation and evaluation in smart grids," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 273–280, 2016.

[38] M. S. Branicky, V. Liberatore, and S. M. Phillips, "Networked control system co-simulation for co-design," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 4, pp. 3341–3346 vol.4, 2003.

[39] X. Tong, "The co-simulation extending for wide-area communication networks in power system," in *2010 Asia-Pacific Power and Energy Engineering Conference*, pp. 1–4, 2010.

[40] M. Ni, Y. Xue, H. Tong, and M. Li, "A cyber physical power system co-simulation platform," in *2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pp. 1–5, 2018.

[41] C. Steinbrink, M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. P. Ramírez Acosta, T. Raub, J. S. Schwarz, S. Stark, *et al.*, "Cpes testing with mosaik: Co-simulation planning, execution and analysis," *Applied Sciences*, vol. 9, no. 5, p. 923, 2019.

[42] OFFIS - Mosaik consortium. `https://mosaik.offis.de/`. [Online; accessed 26-March-2021].

[43] E. de Souza, O. Ardakanian, and I. Nikolaidis, "A co-simulation platform for evaluating cyber security and control applications in the smart grid," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2020.

[44] Wiktionnaire, "hardware-in-the-loop — wiktionnaire." `https://fr.wiktionary.org/w/index.php?title=hardware-in-the-loop&oldid=28611443`, 2020. [Online; accessed 27-March-2021].

[45] Wikipedia contributors, "Hardware-in-the-loop simulation — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Hardware-in-the-loop_simulation&oldid=987981492`, 2020. [Online; accessed 27-March-2021].

[46] B. Jablkowski, O. Spinczyk, M. Kuech, and C. Rehtanz, "A hardware-in-the-loop co-simulation architecture for power system applications in virtual execution environments," in *2014 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pp. 1–6, 2014.

[47] M. Paolone, M. Pignati, P. Romano, S. Sarri, L. Zanni, and R. Cherkaoui, "A hardware-in-the-loop test platform for the real-time state estimation of active distribution networks using phasor measurement units," in *Proc. Cigré SC6 Colloquium*, p. 42, 2013.

[48] S. K. Khaitan, J. D. McCalley, and C. C. Liu, *Cyber physical systems approach to smart electric power grid*. Springer, 2015.

[49] T. C. Nägele, *CoHLA: Rapid Co-simulation Construction*. PhD thesis, [Sl: sn], 2020.

[50] "Ieee standard for modeling and simulation (m amp;s) high level architecture (hla)– framework and rules," *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pp. 1–38, 2010.

[51] Wikipedia contributors, "High level architecture — wikipédia, l'encyclopédie libre." `http://fr.wikipedia.org/w/index.php?title=High_Level_Architecture&oldid=166577981`, 2020. [Online; accessed 27-March-2021].

[52] Wikipedia contributors, "High level architecture — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=High_Level_Architecture&oldid=999126073`, 2021. [Online; accessed 27-March-2021].

[53] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, "Distributed simulation of heterogeneous and real-time systems," in *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*, pp. 55–62, 2013.

[54] H. Georg, S. C. Müller, C. Rehtanz, and C. Wietfeld, "Analyzing cyber-physical energy systems:the inspire cosimulation of power and ict systems using hla," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2364–2373, 2014.

[55] S. Straßburger, "On the hla-based coupling of simulation tools," 1999.

[56] Mathias Fröhlich, "SourceForge - OpenRTI." `https://sourceforge.net/p/openrti/wiki/Home/`, 2021. [Online; accessed 6-April-2021].

[57] FMI consortium. `https://fmi-standard.org/`. [Online; accessed 28-March-2021].

[58] "Bitbucket - DACCOSIM." `https://bitbucket.org/simulage/daccosim/wiki/Home`, 2020. [Online; accessed 06-April-2021].

[59] V. Galtier, S. Vialle, C. Dad, J.-P. Tavella, J.-P. Lam-Yee-Mui, and G. Plessis, "FMI-based distributed multi-simulation with DACCOSIM," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 39–46, 2015.

[60] J. Evora-Gomez, J. Cabrera, j.-p. Tavella, S. Vialle, E. Kremers, and L. Frayssinet, "Daccosim ng: co-simulation made simpler and faster," pp. 785–794, 02 2019.

[61] T. S. Nouidui, J. Coignard, C. Gehbauer, M. Wetter, J.-Y. Joo, and E. Vrettos, "CyDER–an FMI-based co-simulation platform for distributed energy resources," *Journal of Building Performance Simulation*, vol. 12, no. 5, pp. 566–579, 2019.

[62] M. U. Awais, W. Gawlik, G. De-Cillia, and P. Palensky, "Hybrid simulation using SAHISim framework," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 3, no. 9, 2016.

[63] B. Camus, T. Paris, J. Vaubourg, Y. Presse, C. Bourjot, L. Ciarletta, and V. Chevrier, "Co-simulation of cyber-physical systems using a devs wrapping strategy in the mecsyco middleware," *SIMULATION*, vol. 94, no. 12, pp. 1099–1127, 2018.

[64] S. Schütte, S. Scherfke, and M. 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)*, pp. 55–60, IEEE, 2011.

[65] Team SimPy. `https://simpy.readthedocs.io/en/latest/`. [Online; accessed 26-March-2021].

[66] A. van der Meer, P. Palensky, K. Heussen, D. Morales Bondy, O. Gehrke, C. Steinbrinki, M. Blanki, S. Lehnhoff, E. Widl, C. Moyo, T. Strasser, V. Nguyen, N. Akroud, M. Syed, A. Emhemed, S. Rohjans, R. Brandl, and A. Khavari, "Cyber-physical energy systems modeling, test specification, and co-simulation based testing," in *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pp. 1–9, 2017.

[67] "ERIGrid - H2020 Reasearch Infrastructure Project." `https://erigrid.eu/`, 2021. [Online; accessed 01-May-2021].

[68] "LarGo! - Large-Scale Smart Grid Application Roll-Out." `http://www.largo-project.eu/`, 2021. [Online; accessed 30-April-2021].

[69] F. Kintzler, T. Gawron-Deutsch, S. Cejka, J. Schulte, M. Uslar, E. M. Veith, E. Piatkowska, P. Smith, F. Kupzog, H. Sandberg, M. S. Chong, D. Umsonst, and M. Mittelsdorf, "Large Scale Rollout of Smart Grid Services," in *2018 Global Internet of Things Summit (GIoTS)*, pp. 1–7, 2018.

[70] F. Kintzler, S. Cejka, M. Uslar, E. M. Veith, E. Piatkowska, P. Smith, H. Sandberg, M. S. Chong, D. Umsonst, M. Mittelsdorf, F. Pröstl Andrén, J. Kazmi, C. Gavriluta, S. Balduin, M. Azamat, and J. Schumann, "Deliverable D1.1 – Whitepaper Large-Scale Smart Grid Application Roll-Out," 2020.

[71] "GitHub - ComputerNetworks-UFRGS/ASTORIA." `https://github.com/ComputerNetworks-UFRGS/ASTORIA`, 2018. [Online; accessed 30-April-2021].

[72] Wikipédia, "Système de contrôle et d'acquisition de données — Wikipédia, l'encyclopédie libre," 2020. [En ligne; Page disponible le 29-novembre-2020].

[73] "Ptolemy II Home Page." `https://ptolemy.berkeley.edu/ptolemyII/index.htm`, 2021. [Online; accessed 30-April-2021].

[74] M. Wetter, "Co-simulation of building energy and control systems with the building controls virtual test bed," *Journal of Building Performance Simulation*, vol. 4, no. 3, pp. 185–203, 2011.

[75] S. Chatzivasileiadis, M. Bonvini, J. Matanza, R. Yin, T. S. Nouidui, E. C. Kara, R. Parmar, D. Lorenzetti, M. Wetter, and S. Kiliccote, "Cyber–physical modeling of distributed resources for distribution system operations," *Proceedings of the IEEE*, vol. 104, no. 4, pp. 789–806, 2016.

[76] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, "Fncs: A framework for power system and communication networks co-simulation," in *Proceedings of the Symposium on Theory of Modeling amp; Simulation - DEVS Integrative*, DEVS '14, (San Diego, CA, USA), Society for Computer Simulation International, 2014.

[77] "GitHub - FNCS/fncs: Framework for Network Co-Simulation." `https://github.com/FNCS/fncs`, 2018. [Online; accessed 30-April-2021].

[78] Pacific Northwest National Laboratory, "Introducing FNCS: Framework for Network Co-Simulation."

[79] B. Palmintier, D. Krishnamurthy, P. Top, S. Smith, J. Daily, and J. Fuller, "Design of the HELICS high-performance transmission-distribution-communication-market co-simulation framework," in *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pp. 1–6, 2017.

[80] "GitHub - GridAttackSim: Smart Grid Attack Simulation Framework." `https://github.com/FNCS/fncs`, 2021. [Online; accessed 30-April-2021].

[81] T. Duy Le, A. Anwar, R. Beuran, and S. W. Loke, "Smart Grid Co-Simulation Tools: Review and Cybersecurity Case Study," in *2019 7th International Conference on Smart Grid (icSmartGrid)*, pp. 39–45, 2019.

[82] "MECSYCO Dev." http://mecsyco.com/dev/, 2021. [Online; accessed 01-May-2021].

[83] Wikipedia contributors, "Devs — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=DEVS&oldid=995803653, 2020. [Online; accessed 28-March-2021].

[84] J. Vaubourg, Y. Presse, B. Camus, L. Ciarletta, V. Chevrier, J.-P. Tavella, B. Deneuville, and O. Chilard, "Simulation de smart grids avec MECSYCO," *23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15)*, pp. 217–218, 2015.

[85] "HELICS." https://helics.org/, 2021. [Online; accessed 01-May-2021].

[86] "GitHub - Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS)." https://github.com/GMLC-TDC/HELICS, 2021. [Online; accessed 01-May-2021].

[87] N. Duan, N. Yee, B. Salazar, J.-Y. Joo, E. Stewart, and E. Cortez, "Cybersecurity analysis of distribution grid operation with distributed energy resources via co-simulation," in *2020 IEEE Power Energy Society General Meeting (PESGM)*, pp. 1–5, 2020.

[88] "*adevs*: A Discrete Event system Simulator." https://web.ornl.gov/~nutarojj/adevs/, 2006. [Online; accessed 28-March-2021].

[89] "Thyme: Toolkit for hybrid modeling of electric power systems." https://web.ornl.gov/~nutarojj/thyme/docs/, 2020. [Online; accessed 28-March-2021].

[90] J. Nutaro, P. T. Kuruganti, L. Miller, S. Mullen, and M. Shankar, "Integrated hybrid-simulation of electric power and communications systems," in *2007 IEEE Power Engineering Society General Meeting*, pp. 1–8, 2007.

[91] J. Nutaro, "Designing power system simulators for the smart grid: Combining controls, communications, and electro-mechanical dynamics," in *2011 IEEE Power and Energy Society General Meeting*, pp. 1–5, IEEE, 2011.

[92] "CoHLA." https://cohla.nl/intro/, 2019. [Online; accessed 02-May-2021].

[93] V. Liberatore and A. Al-Hammouri, "Smart grid communication and co-simulation," in *IEEE 2011 EnergyTech*, pp. 1–5, 2011.

[94] Wikipédia, "Modelica — wikipédia, l'encyclopédie libre," 2020. [En ligne; Page disponible le 20-août-2020].

[95] D. Bhor, K. Angappan, and K. M. Sivalingam, "A co-simulation framework for smart grid wide-area monitoring networks," in *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–8, 2014.

[96] M. H. R. Centre, "Pscad." https://www.pscad.com. [Online; accessed 30-March-2021].

[97] G. E. CONSULTING, "Pslf." https://www.geenergyconsulting.com/practice-area/software-products/pslf, 2021. [Online; accessed 10-March-2021].

[98] D. GmbH, "Powerfactory - digsilent." https://www.digsilent.de/fr/powerfactory.html. [Online; accessed 30-March-2021].

[99] Siemens, "PSS®SINCAL - simulation software for analysis and planning of electric and pipe networks." https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/pss-software/pss-sincal.html. [Online; accessed 30-March-2021].

[100] Siemens, "PSS®E - high-performance transmission planning and analysis software." https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/pss-software/pss-e.html. [Online; accessed 30-March-2021].

[101] Electricité de France, Hydro-Québec et RTE Réseau de transport d'électricité, "Emtp." https://www.emtp.com/. [Online; accessed 30-March-2021].

[102] "Powerworld simulator." https://www.powerworld.com/. [Online; accessed 30-March-2021].

[103] "pandapower." http://www.pandapower.org/, 2021. [Online; accessed 01-June-2021].

[104] "Power system monitoring simulation - etap." https://etap.com/packages/monitoring-simulation. [Online; accessed 30-March-2021].

[105] "CYME - Analyse de réseaux de distribution CYMDIST." https://www.cyme.com/fr/software/cymdist/, 2020. [Online; accessed 01-June-2021].

[106] "Tractebel Engineering GDF Suez - Eurostag The reference for Power System Dynamic Simulation." http://www.eurostag.be/en/products/eurostag/the-reference-power-system-dynamic-simulation/. [Online; accessed 01-June-2021].

[107] "EPRI | Smart Grid Resource Center - Simulation Tool : OpenDSS." https://smartgrid.epri.com/SimulationTool.aspx, 2020. [Online; accessed 01-June-2021].

[108] "GitHub - ObjectStab: A Modelica Library for Power System Stability Studies." https://github.com/modelica-3rdparty/ObjectStab, 2015. [Online; accessed 01-June-2021].

[109] "GitHub - GridLAB-D." https://github.com/gridlab-d, 2021. [Online; accessed 30-April-2021].

[110] M. H. R. Centre, "Emtdc userguide." https://www.pscad.com/uploads/ck/files/EMTDC%20Users%20Guide%20V4_6_0.pdf. [Online; accessed 30-March-2021].

[111] K. Hopkinson, "Epochs." https://www.cs.cornell.edu/hopkik/epochs.htm, 2004. [Online; accessed 10-March-2021].

[112] M. Fazeli, G. M. Asher, C. Klumpner, L. Yao, and M. Bazargan, "Novel integration of wind generator-energy storage systems within microgrids," *IEEE Transactions on Smart Grid*, vol. 3, no. 2, pp. 728–737, 2012.

[113] Chao Luo, Jun Yang, Yuanzhang Sun, Mao Cai, Jing Wu, and Liangzhu Xu, "A network based protection scheme of distribution system," in *2012 IEEE International Conference on Power System Technology (POWERCON)*, pp. 1–6, 2012.

[114] H. Mahmood and J. Jiang, "Modeling and control system design of a grid connected vsc considering the effect of the interface transformer type," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 122–134, 2012.

[115] "OPC Foundation - What is OPC?." `https://opcfoundation.org/about/what-is-opc/`, 2021. [Online; accessed 01-June-2021].

[116] F. Andrén, M. Stifter, T. Strasser, and D. Burnier de Castro, "Framework for co-ordinated simulation of power networks and components in smart grids using common communication protocols," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2700–2705, 2011.

[117] F. Coroiu, C. Velicescu, and C. Barbulescu, "Probabilistic and deterministic load flows methods in power systems reliability estimation," in *2011 IEEE EUROCON - International Conference on Computer as a Tool*, pp. 1–4, 2011.

[118] Xinhe Chen, Wei Pei, and Xisheng Tang, "Transient stability analyses of micro-grids with multiple distributed generations," in *2010 International Conference on Power System Technology*, pp. 1–8, 2010.

[119] T. I. Chant, G. Shafiullah, A. M. T. Oo, and B. E. Harvey, "Impacts of increased photovoltaic panel utilisation on utility grid operations - a case study," in *2011 IEEE PES Innovative Smart Grid Technologies*, pp. 1–7, 2011.

[120] J. Bergmann, C. Glomb, J. Gotz, J. Heuer, R. Kuntschke, and M. Winter, "Scalability of smart grid protocols: Protocols and their simulative evaluation for massively distributed ders," *2010 First IEEE International Conference on Smart Grid Communications*, pp. 131–136, 2010.

[121] A. M. Mohamad, N. Hashim, N. Hamzah, N. F. Nik Ismail, and M. F. Abdul Latip, "Transient stability analysis on sarawak's grid using power system simulator for engineering (pss/e)," in *2011 IEEE Symposium on Industrial Electronics and Applications*, pp. 521–526, 2011.

[122] A. Hernandez, P. Eguia, E. Torres, and M. A. Rodriguez, "Dynamic simulation of a sssc for power flow control during transmission network contingencies," in *2011 IEEE Trondheim PowerTech*, pp. 1–6, 2011.

[123] F. Napolitano, A. Borghetti, M. Paolone, and M. Bernardi, "Voltage transient measurements in a distribution network correlated with data from lightning location system and from sequence of events recorders," *Electric Power Systems Research*, vol. 81, no. 2, pp. 237–253, 2011.

[124] "About pandapower - pandapower." `http://www.pandapower.org/about`, 2021. [Online; accessed 01-June-2021].

[125] R. Mehra, N. Bhatt, F. Kazi, and N. M. Singh, "Analysis of pca based compression and denoising of smart grid data under normal and fault conditions," in *2013 IEEE International Conference on Electronics, Computing and Communication Technologies*, pp. 1–6, 2013.

[126] T. Godfrey, S. Mullen, D. W. Griffith, N. Golmie, R. C. Dugan, and C. Rodine, "Modeling smart grid applications with co-simulation," in *2010 First IEEE International Conference on Smart Grid Communications*, pp. 291–296, 2010.

[127] A. Awad, P. Bazan, and R. German, "Sgsim: A simulation framework for smart grid applications," in *2014 IEEE International Energy Conference (ENERGYCON)*, pp. 730–736, 2014.

[128] "Opal-rt - emegasim." `https://www.opal-rt.com/system-emegasim/`, 2019. [Online; accessed 29-March-2021].

[129] "Opal-rt." `https://www.opal-rt.com/`, 2019. [Online; accessed 29-March-2021].

[130] A. Srivastava and N. Schulz, "Applications of a real time digital simulator in power system education and research," 2009.

[131] "Rtds technologies - rtds simulator." `https://www.rtds.com/`, 2021. [Online; accessed 29-March-2021].

[132] M. default, "ns-2." `http://nsnam.sourceforge.net/wiki/index.php/Main_Page`, 2014. [Online; accessed 30-March-2021].

[133] "ns: Change history." `https://www.isi.edu/nsnam/ns/CHANGES.html`. [Online; accessed 30-March-2021].

[134] Wikipedia contributors, "Ns (simulator) — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Ns_(simulator)&oldid=1005533610`, 2021. [Online; accessed 31-March-2021].

[135] Z. Pan, Q. Xu, C. Chen, and X. Guan, "Ns3-matlab co-simulator for cyber-physical systems in smart grid," in *2016 35th Chinese Control Conference (CCC)*, pp. 9831–9836, 2016.

[136] B. M. Kelley, P. Top, S. G. Smith, C. S. Woodward, and L. 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)*, pp. 1–6, 2015.

[137] Wikipédia, "Network Simulator — Wikipédia, l'encyclopédie libre," 2020. [En ligne; Page disponible le 16-septembre-2020].

[138] "OMNeT++ Discrete Event Simulator." `https://omnetpp.org/`, 2021. [Online; accessed 02-June-2021].

[139] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pp. 1–10, 2008.

[140] "OMNEST - High-Performance Simulation for All Kind of Networks." `https://omnest.com/`, 2021. [Online; accessed 02-June-2021].

[141] Wikipédia, "Cisco Systems — Wikipédia, l'encyclopédie libre," 2021. [En ligne; Page disponible le 20-mai-2021].

[142] E. Tebekaemi and D. Wijesekera, "Designing an iec 61850 based power distribution substation simulation/emulation testbed for cyber-physical security studies," in *Proceedings of the First International Conference on Cyber-Technologies and Cyber-Systems*, pp. 41–49, 2016.

[143] "NetSim - Network Simulator Emulator | Home." `https://www.tetcos.com/`, 2021. [Online; accessed 02-June-2021].

[144] "NeSSi²." `http://www.nessi2.de/`, 2013. [Online; accessed 02-June-2021].

[145] J. Chinnow, K. Bsufka, A. Schmidt, R. Bye, A. Camtepe, and S. Albayrak, "A simulation framework for smart meter security evaluation," in *2011 IEEE International Conference on Smart Measurements of Future Grids (SMFG) Proceedings*, pp. 1–9, 2011.

[146] T. Konnerth, J. Chinnow, S. Kaiser, D. Grunewald, K. Bsufka, and S. Albayrak, "Integration of simulations and MAS for smart grid management systems," in *Proceedings of the 3textsuperscriptrd International Workshop on Agent Technologies for Energy Systems (ATES 2012), Valencia, Spain*, pp. 51–58, 2012.

[147] H. Georg, S. C. Müller, N. Dorsch, C. Rehtanz, and C. Wietfeld, "Inspire: Integrated co-simulation of power and ict systems for real-time evaluation," in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 576–581, 2013.

[148] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri, "A testbed for analyzing security of scada control systems (tasscs)," in *ISGT 2011*, pp. 1–7, 2011.

[149] Wikipedia contributors, "Qualnet — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=QualNet&oldid=1001055603`, 2021. [Online; accessed 31-March-2021].

[150] R. Gupta, "Substation automation using iec61850 standard," in *Fifteenth National Power Systems Conference (NPSC), IIT Bombay*, pp. 462–466, 2008.

[151] Circuit Globe, "Potential Transformer (PT)." `https://circuitglobe.com/potential-transformer-pt.html`. [Online; accessed 05-June-2021].

[152] "mosaik / mosaik-pandapower · GitLab." `https://gitlab.com/mosaik/mosaik-pandapower`. [Online; last development March 2021].

[153] "mosaik / mosaik-householdsim · GitLab." `https://gitlab.com/mosaik/mosaik-householdsim`. [Online; last development September 2019].

[154] Wikipédia, "Hierarchical Data Format — Wikipédia, l'encyclopédie libre," 2020. [En ligne; Page disponible le 8-mai-2020].

[155] "mosaik / mosaik-hdf5 · GitLab." `https://gitlab.com/mosaik/mosaik-hdf5`. [Online; last development June 2019].

[156] "mosaik / mosaik-demo · GitLab." `https://gitlab.com/mosaik/mosaik-demo`. [Online; last development March 2021].

[157] "Read the Docs - Predefined Controllers - pandapower 2.6.0 documentation." `https://pandapower.readthedocs.io/en/v2.6.0/control/controller.html#discrete-tap-control`. [Online; accessed 29-May-2021].

[158] "mosaik / mosaik-communication · GitLab." https://gitlab.com/mosaik/mosaik-communication. [Online; last development March 2021].

[159] "Read the Docs - Networks - pandapower 2.6.0 documentation." https://pandapower.readthedocs.io/en/v2.6.0/networks.html. [Online; accessed 11-May-2021].

[160] "GitHub - co_sim_platform : A smart grid co-simulation platform based on mosaik." https://github.com/sustainable-computing/co_sim_platform, 2021. [Online; accessed 30-April-2021].