

Chapter 3

Simulation-based Assessment Methods

A. A. van der Meer, R. Bhandia, P. Palensky, M. Cvetković, E. Widl, V.H. Nguyen, Q.T. Tran and K. Heussen

3.1 Introduction to Smart Grid Modelling and Simulation

In general, smart grids can be considered as the application of various types of automation and control for the operation of energy technology, with a focus on electrical power engineering. Examples include the application of distributed automation in substations, smart metering of domestic consumers, and wide-area protection mechanisms. Such technology allows the energy systems to be operated and controlled more optimally and to be pushed to their design boundaries. Notwithstanding these advantages, these concepts heavily rely on ICT structures, which form the glue between the physical domain (e.g., energy systems and their components) and the control and automation domain (e.g., decision-making devices, overarching logic and algorithms). Together, these domains constitute the concept of cyber-physical energy systems (CPES).

The domain coupling challenges of CPES are evident. The overall system exhibits multi-time scale (transients versus market decisions) interactions, multi-size (decentralised measurements, wide-area protection) properties, and heterogeneous (physical versus discrete events) behaviour. In order to assess the operation, security, and reliability of CPES, the common way of testing and validation for smart energy components shall be reconsidered. Eventually, lab-based approaches to test, validate

Arjen A. van der Meer, Rishabh Bhandia, Peter Palensky, Miloš Cvetković
Delft University of Technology, Delft, The Netherlands e-mail: a.a.vandermeer@tudelft.nl

Edmund Widl
AIT Austrian Institute of Technology, Vienna, Austria

Van Hoa Nguyen, Quoc Tuan Tran
Université Grenoble Alpes, INES, Le Bourget du Lac, France and CEA, LITEN, Le Bourget du Lac, France

Kai Heussen
Technical University of Denmark, Roskilde, Denmark

and roll-out new concepts must be able to capture the cross-domain interactions the system or component under test will be subject to.

One of the steps that need to be taken to achieve this is analysis, modelling, and simulation of cross-domain interactions. Fig. 3.1 a) and b) show for example of how domain-specific models (e.g., physical and ICT) are usually simulated by dedicated tools with specialised solvers. Cross domain interactions can be included by attempting to model the entire system under test in a general-purpose simulation tool like Simulink or OpenModelica (i.e., Fig. 3.1). This approach has the advantage of maintaining the entire model into one simulation tool. A common downside is that such models commonly scale badly in size and phenomena addressed. Another method is to include the model of the 'alien' domain (say model B of Fig. 3.1 d)) into a specialised tool and subsequently make this model compatible with the applied solver. This is usually done when it is assumed justifiable to simplify parts of the overall model to make it suitable for a single-domain tool.

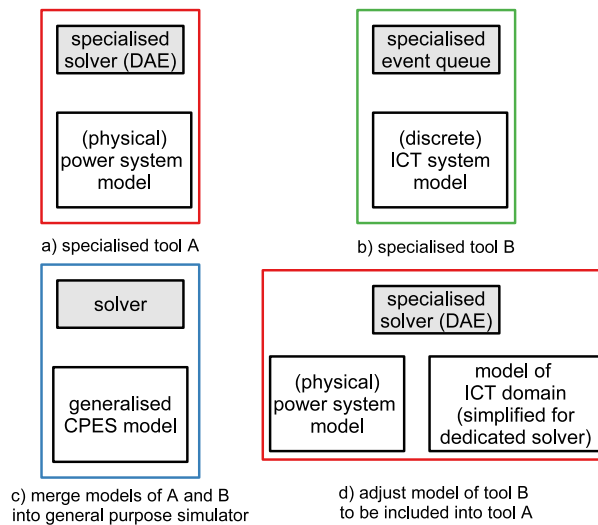


Fig. 3.1 Overview of simulation based assessment of CPES. Domain-specific approaches (i.e., a and b), and multi-domain simulations (i.e., c and d).

Fig. 3.1 often lead to a suboptimal trade-off between simulation efficiency (speed) and accuracy of the phenomena of interest. This chapter will focus on the simulation aspects of this challenge and more specifically on one particular method to deal with this: coupled simulations, also referred to as co-simulations [7]. As an assessment approach, co-simulation offers key advantages for the simulation of cyber-physical systems-of-systems:

1. *Modularity* - Co-simulation allows to represent (parts of) sub-systems with the most appropriate tool available. This encourages a modular representation of the

system under test, with clean semantic and functional model boundaries along the real-world domain borders.

2. *Hierarchical composition* as a feature of system-of-systems architectures, is supported in co-simulation by the modular approach, where a hierarchical modelling strategy allows to switch out abstracted functional representations with explicit models of system layers (e.g., abstracted ICT layer: point-to-point information exchange, detailed: explicit transport layer model).

In the following, the basic concepts of co-simulations will be explained. Then the available standardized approaches to set up a co-simulation, such as the high-level architecture and the functional mock-up interface will be introduced. Finally, a survey of scaling aspects in terms of co-simulation of CPES follows and an example implementation is discussed, which concerns coupling a power system simulation to a general-purpose simulation.

3.2 Co-Simulation based Assessment

3.2.1 Introduction to Co-Simulation, Goals, and Challenges

The term co-simulation is typically used, when two or more models are used in one simulation. (Real) Co-simulation happens, when these two or more separate models are executed concurrently and if their variables or states depend on each other (i.e., option c in Fig. 3.2). These simulators have to synchronize with each other periodically. Sometimes embedding a model into another one and executing them with just one simulator (i.e. numerical solver) is called co-simulation with model exchange. If one simulation component is just uni-directionally using data (e.g., time series) from another simulation we speak of sequential simulation.

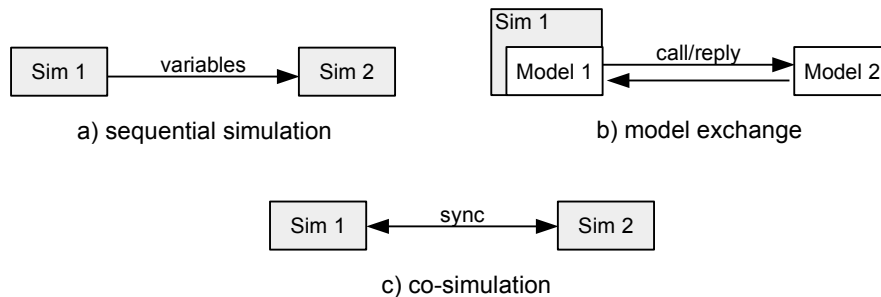


Fig. 3.2 Variants of coupled simulations.

Fig. 3.3 shows the time sequence and data flow of two simulators that are coupled via a master algorithm. The master must have some possibility to start and stop

the simulators, ideally in an on-the-fly fashion that does not require re-initialization of the states. The choice of synchronization steps is usually up to the co-simulation engineer. If both simulators have fixed time steps, and if these time steps are multiples of each other, the synchronization becomes easy. If, however, the time steps are totally independent of each other the master might have to interpolate variables that are exchanged between two-time steps. This situation is shown in Fig. 3.3 and might require the master to roll-back simulators from time to time if possible and required.

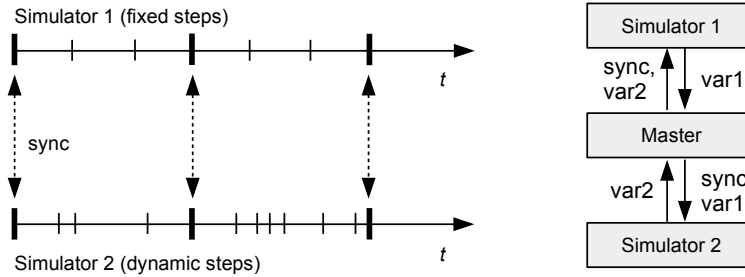


Fig. 3.3 A master algorithm synchronizes the simulators and passes on shared variables.

The reasons why co-simulation is often used are usually pragmatic and solution oriented:

- Existing legacy models can be used with new models. Often it is not feasible to re-implement existing models in the simulator of choice with the given resources (such as shown in Fig. 3.2 b)).
- Specialized simulators can be used for parts of a multi-disciplinary problem. By that, models of one sub-model (e.g., discrete events) do not have to be badly "imitated" in the simulator of another sub-problem (e.g., continuous dynamics). The specialized simulators usually have a better (and maybe even validated) model library and a tailored work-flow and user-interface for their particular domain.
- The simulation study can have multiple foci. Unlike in the case of a standard (monolithic) simulation, there is no need to simplify sub-problems. Each sub-problem (may it be mechanical, thermal, electric, economic, etc.) can be modelled in all detail since it runs (and can even be tested) in its own specialized environment.

These advantages have to be contrasted with a number of disadvantages, too:

- Models in different modelling environments need to be maintained. This involves multiple modelling languages, simulation project files, and software licenses. This also means that the staff, doing the simulation, needs to be educated in all these tools - plus the co-simulation environment! A high level of versioning and documentation discipline is required to achieve a sustainable way of working with that.

- Experience shows, that co-simulation is slow. Although its perfect suitability for parallel computation would suggest speed gains, it is the synchronization of simulations (that were often not designed to be synchronized) that slows things down. Some legacy simulators process licensing information when they are stopped and restarted or require a fresh initialization, which of course grinds down performance if frequent synchronization is needed.
- Error propagation and estimation of co-simulation is poorly understood. The choice of simulation step sizes or synchronization points is therefore not trivial and is still subject of technical developments.

Interfacing with the master can be done via various APIs (application program interfaces), one that received broad industry support is called FMI: the Functional Mockup Interface. It is an open standard, based on a C-interface that offers the specification of required functions such as start, stop, step, synchronization, variable exchange, etc.

The master algorithm itself is in its core often very simple but associated functionality (such as scenario handling, data logging, distributed computing, etc.) can be quite complex. Again, there are a few popular master platforms, two of them being HLA (the high-level architecture) and mosaik.

Once the master and the simulators are set up, the work flow is very much as a standard simulation-based analysis: Scenarios are generated (e.g., parameter sweeps, etc.), and an optimizer or engineer runs these scenarios in a number of simulations until the expected result is found.

3.2.2 Current Co-simulation Standards and their Functionality

The High Level Architecture (HLA) was originally developed under the umbrella of the Department of Defense of the USA in order to serve its high demands for a versatile simulation environment. The development was initiated in the early nineties, and the current HLA version is standardized under HLA 1516-2010 (known also as HLA Evolved) [1]. This standard does not focus on the implementation of the co-simulation master, but instead, establishes the list of services that must be provided by the master (in HLA terms called Run-Time Infrastructure (RTI)). Some of the greatest advantages of HLA are its versatility and configurability, while at the same time, these features amount to a steep learning curve for the co-simulation engineer.

The Functional Mockup Interface (FMI) was created to ease model exchange between vendors of various components assembling larger physical systems (one such example is automotive industry). Therefore, its primary focus is on model encapsulation (within so-called Functional Mockup Unit (FMU)) and its current standard, FMI 2.0, provides a comprehensive interface for model engagement (such as model evaluation, Jacobian retrieval, etc.) [4]. As the second step in its evolution, FMI was enhanced with a co-simulation interface (such as starting, stopping, stepping of the models). The current standard anticipates packaging of FMUs with and without internal solvers. If packaged with internal solvers, the FMUs can be directly included

in co-simulation. Otherwise an external solver must be engaged to step the model within FMU.

FMI and HLA are complementary in nature, since FMI focuses on the engagement of models, while HLA focuses on the master services [5]. Today, many engineering simulation tools allow to export the models as FMUs, which improves the breadth of co-simulation scope.

Finally, *mosaik* was created with a particular intention to serve as a smart grid co-simulation framework, and as such, it is largely in tune with energy system applications [8]. In contrast to the previously mentioned standards, *mosaik* is a direct implementation of a master algorithm, and not a standard per se. It is written in Python, based on a discrete event scheduler and is capable of FMU integration. Besides FMU integration, it also provides interfaces for several common tools in the energy system realm (such as DigSilent PowerFactory, PandaPower, etc.). Since its primary user group are energy engineers, the elaborate co-simulation settings are greatly simplified, which represents *mosaik*'s greatest advantage. A comparison of *mosaik* and HLA for a co-simulation of a power system control action is performed in [9].

3.3 Co-simulation Framework for Smart-Grid Assessment

3.3.1 Co-simulation Interfaces based on FMI

In order to accurately simulate Smart Grids, the interaction between the domains of *electrical power systems*, *communication* and *automation and control* is of crucial importance. As a proof-of-concept, co-simulation interfaces based on the FMI standard have been developed for selected state-of-the-art tools, examples of which are described in the following.

3.3.1.1 Power System Simulation with PowerFactory

*DIGSILENT PowerFactory*¹ is a commercial tool for power system design and analyses. PowerFactory does not officially provide an FMI-compliant co-simulation interface. However, it provides an API that enables basic interactions with simulation models at run-time like setting/retrieving variables and calculating power flows.

Furthermore, PowerFactory provides the possibility to issue so-called *events* during time-domain simulations (more specifically, *RMS simulations* in PowerFactory) that can change the system state at a specified point in simulation time. This mechanism has been utilized to enable a dynamic interaction with simulation models at

¹ DIGSILENT PowerFactory, <http://www.digsilent.com>, accessed October 18, 2019.

run-time. It is suited for co-simulation and has been integrated into a stand-alone FMU exporter tool².

3.3.1.2 Communication Network Simulation with ns-3

In recent years, *ns-3* has become very popular in the network simulation community. *ns-3* is a highly flexible simulation package, which allows programmers to add new attributes without modifying the core of the source code, or having to deal with a specific, restricted and complex API. The default version of *ns-3* comes with an extensive library of models, which can be used to describe the components and other aspects of communication networks (e.g., devices, channels, interfaces, protocols).

A dedicated *ns-3* package called *fmi-export* has been developed, which provides all functionality needed for creating an FMU for C-Simulation from a user-defined *ns-3* application (typically referred to as *script*). An FMU created with the help of this package implements a tool coupling mechanism that allows to control the execution of the *ns-3* simulator and to establish a connection for data exchange during run-time.

3.3.1.3 Control Simulation with MATLAB

Despite the popularity and widespread use of the numerical computing environment *MATLAB*, there is so far only comparably little support within the context of FMI. For instance, the *Modelon FMI Toolbox*³ and the *FMI Kit for Simulink*⁴ offer the export of Simulink models as FMUs for Model Exchange, but so far there is no tool available that allows to provide *MATLAB*'s full functionality via an FMI-compliant co-simulation interface.

Therefore, the *FMI++ MATLAB Toolbox*⁵ has been implemented that provides two components: a *front-end component* to be used by the co-simulation master and a *back-end component* to be used by *MATLAB*. The corresponding interfaces are tailored to suit the requirements of the FMI specification and they implement the necessary functionality required for a master-slave concept, i.e., synchronization mechanisms and exchange of data.

² The FMI++ PowerFactory FMU Export Utility, <http://powerfactory-fmu.sourceforge.net>, accessed October 18, 2019.

³ FMI Toolbox for MATLAB/Simulink, <https://www.modelon.com/products-services/\\modelon-deployment-suite/fmi-toolbox>, accessed October 18, 2019.

⁴ FMI Kit, <https://www.3ds.com/products-services/catia/products/dymola/fmi/>, accessed October 18, 2019.

⁵ The FMI++ MATLAB Toolbox, <http://matlab-fmu.sourceforge.net>, accessed October 18, 2019.

3.3.2 Mosaik for Scenario Development and Simulation Orchestration

The *mosaik*⁶ framework is an easy-to-deploy software package that facilitates the integration of new simulators as well as the creation of co-simulation experiments. This is achieved via a lightweight software core based purely on Python, a special Component-API for simulator integration, and a Scenario-API for flexible simulator coupling. The mosaik framework is still under active development and new features are being introduced based on activities within the smart grid testing and validation community.

3.3.2.1 FMI Support

As an example, the *FMI++ Python Interface*⁷ and the mosaik framework have been successfully combined for the co-simulation of FMUs. Several examples of importing FMUs have been implemented using the FMI++ Python Interface to interact with the FMU and mosaik's high-level component API to integrate it into the co-simulation. For this, especially the functionality for conveniently handling FMUs was extensively used, such as extracting the FMU, parsing its model description or the ability to refer to input/output variables by name (rather than the numerical value reference associated to each variable).

3.3.2.2 Handling of Cyclic Dependencies

The term *cyclic dependencies* refers to a co-simulation setup in which two (or more) simulators require data from each other to advance their state in time (i.e., Fig. 3.2 c)). These data dependencies may lead to deadlocks with all simulators waiting for data from each other, halting the whole simulation process. Therefore, proper handling of these cyclic dependencies is one of the most crucial tasks in co-simulation. This is especially true in the case of Smart Grid applications, which typically involve feedback loops and a strong physical coupling between the individual components and subsystems.

The co-simulation framework mosaik has been developed with a strong focus on flexibility in terms of configuring the connected simulators. Accordingly, the scheduling algorithm of mosaik is designed in a way to allow integration of any number of simulators. Furthermore, all integrated simulators may display different step sizes and even vary their step size over time. In order to guarantee the absence of deadlocks for any given setup, the handling of cyclic dependencies in mosaik has so far had some limiting characteristics. In particular, using mosaik's intuitive

⁶ The mosaik Smart Grid co-simulation framework, <http://mosaik.offis.de/>, accessed October 18, 2019.

⁷ The FMI++ Python Interface, <https://pypi.org/project/fmipp/>, accessed October 18, 2019.

3.3.2 Mosaik for Scenario Development and Simulation Orchestration

The *mosaik*⁶ framework is an easy-to-deploy software package that facilitates the integration of new simulators as well as the creation of co-simulation experiments. This is achieved via a lightweight software core based purely on Python, a special Component-API for simulator integration, and a Scenario-API for flexible simulator coupling. The mosaik framework is still under active development and new features are being introduced based on activities within the smart grid testing and validation community.

3.3.2.1 FMI Support

As an example, the *FMI++ Python Interface*⁷ and the mosaik framework have been successfully combined for the co-simulation of FMUs. Several examples of importing FMUs have been implemented using the FMI++ Python Interface to interact with the FMU and mosaik's high-level component API to integrate it into the co-simulation. For this, especially the functionality for conveniently handling FMUs was extensively used, such as extracting the FMU, parsing its model description or the ability to refer to input/output variables by name (rather than the numerical value reference associated to each variable).

3.3.2.2 Handling of Cyclic Dependencies

The term *cyclic dependencies* refers to a co-simulation setup in which two (or more) simulators require data from each other to advance their state in time (i.e., Fig. 3.2 c)). These data dependencies may lead to deadlocks with all simulators waiting for data from each other, halting the whole simulation process. Therefore, proper handling of these cyclic dependencies is one of the most crucial tasks in co-simulation. This is especially true in the case of Smart Grid applications, which typically involve feedback loops and a strong physical coupling between the individual components and subsystems.

The co-simulation framework mosaik has been developed with a strong focus on flexibility in terms of configuring the connected simulators. Accordingly, the scheduling algorithm of mosaik is designed in a way to allow integration of any number of simulators. Furthermore, all integrated simulators may display different step sizes and even vary their step size over time. In order to guarantee the absence of deadlocks for any given setup, the handling of cyclic dependencies in mosaik has so far had some limiting characteristics. In particular, using mosaik's intuitive

⁶ The mosaik Smart Grid co-simulation framework, <http://mosaik.offis.de/>, accessed October 18, 2019.

⁷ The FMI++ Python Interface, <https://pypi.org/project/fmipp/>, accessed October 18, 2019.

connection capabilities to establish cyclic data exchange between two or more simulators has been prohibited. Instead, users had to extend the simulator interfaces to realize cyclic data exchange, which obviously decreases the usability of mosaik for researchers with limited programming experience. Furthermore, the described solution in mosaik only supports serial data exchange schemes.

Recently, the capabilities of mosaik have been extended to allow for higher usability in the handling of cyclic dependencies. The basic idea of this extension is the separation of data exchange into two stages: Simulators may receive data either before they are called to calculate a time step, or after they have calculated so that they store the data for the next time they are called. With this separation, priorities between simulators can be established so that deadlocks are avoided. Fig. 3.4 illustrates different data exchange options between two simulators A and B. Connections for data exchange before calculations are called standard connections since they are part of the typical functionality of mosaik. The newly added connection type is called time-shifted connection since they provide data to simulators that already have been called for calculation.

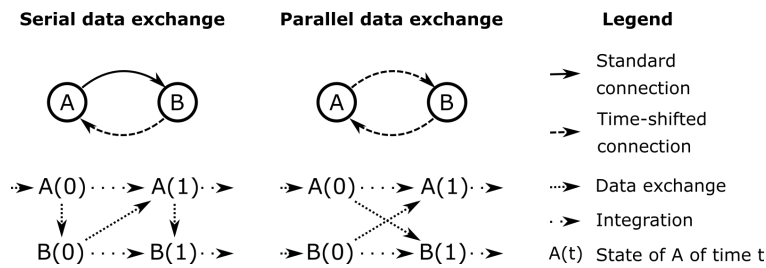


Fig. 3.4 Possible data exchange schemes in mosaik.

Fig. 3.4 shows that standard connections in mosaik provide data to a simulator for its calculation of the current time step while time-shifted connections provide data for the next time step to be calculated. Furthermore, mosaik provides the option to set default input data for the first calculation of a simulator that is addressed by time-shifted connections. In this way, parallel data exchange schemes may also be realized if initial input data can be assigned to each simulator. Overall, the extension of mosaik improves its usability and provides it with the most common options for handling cyclic dependencies in black box co-simulation for smart grid applications.

3.4 Scaling Considerations

The purpose of simulation-based smart grid assessment is often the ability to evaluate the (often non-linear) effects of changes in system parameters on large systems that cannot be established through abstracted analytical models or limited physical experiments with few hardware components.

Scaling-up of established simulation components to a large-scale scenario is conceptually simple in co-simulation: due to the modularity and hierarchical build-up of models, system components can be re-used with alternative parameters and scenario APIs allow scripted scenario configuration and handling.

However, there is a number of non-trivial issues that needs to be considered when planning and developing scale-up simulations, arising from either a) the complexity of system interactions represented, or b) the increasing simulation program scale and complexity [2]. Table 3.1 offers a view on several types of *large-scale phenomena* in energy systems, distinguishing whether these emerge from the physical domain (real-world application) or inaccuracies of the research infrastructure (laboratory or simulation environment).

Table 3.1 Large scale phenomena considered in the context of smart grids

Real world (investigated phenomena)	Physical (Laboratory)	Virtual (Simulation)
Scale in number of nodes and components	number of nodes/ buses/ components	number of equations
Complexity through inter-dependencies across multiple domains	number of domains (power, heat, ICT etc.)	number of simulation tools and instances
complexity through stakeholder interpretations	number of relevant layers (business, information, communication, components etc.)	variety of models of computation (time-continuous, event-driven, stochastic, etc.)
Socio-geographical size	Geographical size	

To distinguish Large Scale System (LSS) phenomena, we characterize them by their effects on system parameters, as presented in Table 3.2. The phenomena are characterized by the observable relation between system input and control parameters—factors in design of experiment (DoE) terms—and the resulting variation of observation parameters (observables, performance metrics, DoE: response variables).

Table 3.2 LSS phenomena characterization chart

Dependency on control parameters	<i>Scale with the system size</i> (linear, logarithmic, exponential, and polynomial) <i>Appear at certain critical system sizes</i> (i.e. phenomenon appears and remains beyond a certain control parameter value) <i>Appear and disappear at certain operational zone</i> of control parameters or parameter combinations
Variation of observation parameters	Extreme values (e.g. performance increase or decay; system failure) Inadvertent oscillations Intermittent performance degradation

In order to consider appropriate assessment methods for the aforementioned categories, two principal scaling approaches can be adopted:

- *Upscaling in terms of system properties (i.e., scale out)*: this method targets phenomena directly related to physically large scale systems. E.g., how does the co-simulation scale with physical system size?
- *Upscaling in terms of simulation and modelling (i.e., scale up)*: this method targets large scale implementations in models and simulation for the validation of smart grids. E.g., how does the co-simulation scale with the number of models and simulations involved?

The co-simulation example demonstrated in the next section introduces a co-simulation that has been subject to upscaling principles: in terms of properties - scale out (rate power, number of wind turbines) and in terms of modelling - scale up (number of FMUs) [3].

3.5 Fault Ride-Through of a Wind Park Example

This section comprises a typical test case in which domains and their co-simulation challenges come together, with a focus is on the evaluation of cyclic dependencies between different models in the context of co-simulation. In the implementation example a standard IEEE 9-bus dynamic test system is modified to contain a Wind Power Plant (WPP), which replaces one of the 3 main generators. A wind park is typically subject to grid connection requirements by the network owner, formulated in grid codes. The Fault ride-through (FRT) capability of WPPs is such a requirement, the assessment of which requires a detailed dynamic simulation of the system, a simulation that encompasses numerous cyclic dependencies between different system components and the general maintenance of synchronism (i.e., transient and frequency stability). As a result, FRT serves as a rigorous test for co-simulation tools and simulation interfaces.

The main models involved in the implementation of the test case include dynamic models of the WPP, converter controller and FRT controller. The standard IEEE-9 bus system was modified to replace the generator at bus 3 by a WPP consisting of full converter interfaced generators. The WPP is connected to the rest of the grid at the point of common coupling (PCC). The PCC is significant since all the important metrics like compliance to appropriate voltage-time profiles during FRT is monitored and evaluated here and forms a legal boundary between the plant assets owner and the grid operator. The controllers developed are embedded in the AC-DC converter controller. A single-line diagram of the experimental setup can be seen in Fig. 3.5).

The WPP is an aggregated version of a medium to large scale onshore wind power park. The WPP is rated at 85MVA which is cumulative rating of 32 wind turbines, each having a power rating of 2.6 MVA. The wind turbines are assumed to be deployed in an 8X4 array distanced by 700m each.

The *converter controller* is designed with two proportional-integral controllers and one overarching current limiter. The controller is a *grid following* vector controller, which also models the reference voltage signals to control the voltages on

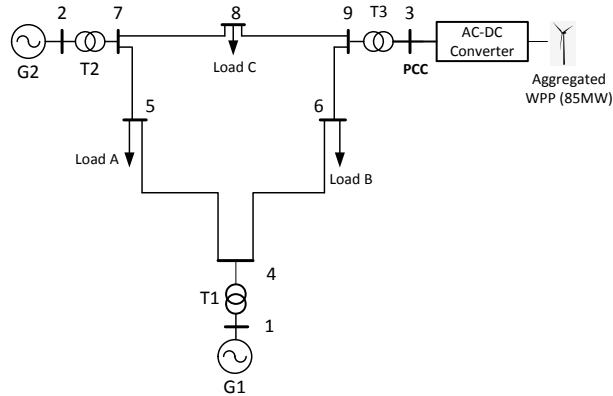


Fig. 3.5 Modified IEEE 9-bus system acting as a test system for co-simulating PowerFactory with Matlab/Simulink using the functional mock-up interface.

both AC and DC side. The q -axis controller regulates the voltage magnitude of the PCC, whereas the d -axis controller maintains the active power reference [6].

The *FRT controller* acts on the top of the converter controller as a discrete finite state machine. It monitors the voltages on both AC and DC sides to sense fault conditions and shifts from normal control mode to FRT control, post-FRT mode, and back to normal control accordingly. During the FRT mode, the FRT controller increases the reactive current infeed and blocks active power flow to address the voltage dip. In post-FRT mode, the FRT controller sets a maximum ramping rate for restoring the active current reference back to pre-fault conditions.

3.5.1 Experiment Setup and Objectives

The co-simulation is set up as follows. The AC grid including the wind part array is modelled and simulated in DIGSILENT PowerFactory. Inside PowerFactory, the wind turbine is modelled as a Norton equivalent source, the current injection of which can vary in time and is provided by the parameter event functionality as discussed in Section 3.3.1.1. This can be considered as a proxy model of the actual converter dynamics by the converter controller and FRT controllers, which are developed in Simulink and Matlab respectively. During runtime of the co-simulation, all FMUs are synchronised using fixed macro time step-sizes of 10 ms.

A 3-phase short circuit event is simulated to study the FRT capability. The co-simulation is orchestrated by a Python script, which uses the FMI++ toolbox. Eventually the overall system under test is split into three FMUs. The three FMUs being: One FMU for the entire power system model in PowerFactory, one for the FRT controller and one for the converter controller. The arrangement of the FMUs and the co-simulation orchestration by Python and FMI++ is shown in Fig. 3.6.

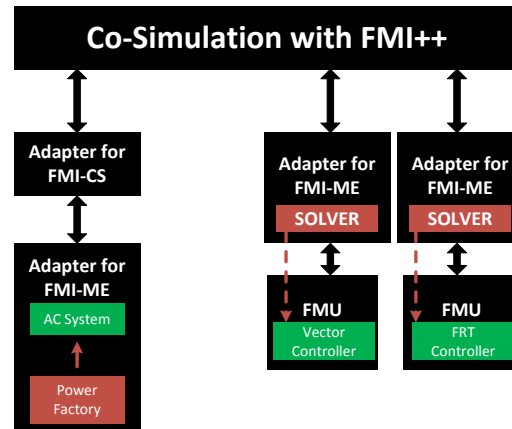


Fig. 3.6 Co-simulation Setup. On top the Python script using FMI++ to interface with the functional mockup units below. Left the FMU of PowerFactory based on FMI for co-simulation, in the centre the FMU of the converter (vector) controller based on FMI for model exchange. On the right the FMU of the fault ride-through controller based on FMI for model exchange. Both have an encapsulated dedicated numerical solver.

The simulation itself is centred around the dynamic response of the WPP and the IEEE 9-bus system that is subject to a self-cleared 180 ms 3-phase short circuit starting at $t = 1$ s (Bus 6 of Fig 3.5). The main objective is to study the FRT capability and reactive power control of the WPP. These ensure that the voltage at the PCC does not dip beyond the FRT voltage versus time profile, and quickly ramp the voltage to pre-fault levels after fault clearance (i.e., grid code compliance). During the event, the WPP shall remain connected to the grid. Adherence to these conditions during the simulations will certify that the co-simulation tools and interfaces have performed at their expected levels. In order to validate the co-simulation, a reference monolithic simulation was conducted in PowerFactory, too, in which standard dynamic models and dedicated DSL for the converter controls have been employed to duplicate the model specification in Matlab/Simulink.

3.5.2 Results

Figure 3.7 and 3.8 show the voltage magnitude at the PCC and active power through the PCC respectively. Taking the monolithic (PowerFactory only) simulation as a reference, it can be seen that the voltage sag experienced at the PCC is around 50 % of nominal, which, quickly restores after fault clearance and swings back to values around nominal seconds after. This fast restoration is owing to the relatively strong grid as well as the voltage-dependent reactive current injection during the voltage

dip. The presence of the active power recovery rate, engaged by the FRT controller, can also be clearly distinguished.

Despite the rather tight coupling between the submodels of the co-simulation, the dynamics around the PCC (red solid line) follow the reference simulation generally well. During fault ignition and clearance, a small discrepancy can be observed, particularly in the voltage magnitude, which can be attributed to the numerical oscillations caused by the serial data exchange protocol (see Fig. 3.4) that is applied in the master algorithm. Especially the active power, which can be considered a *flow* variable, traces the monolithic simulation very similarly. This enhances the validity of the co-simulation as a whole.

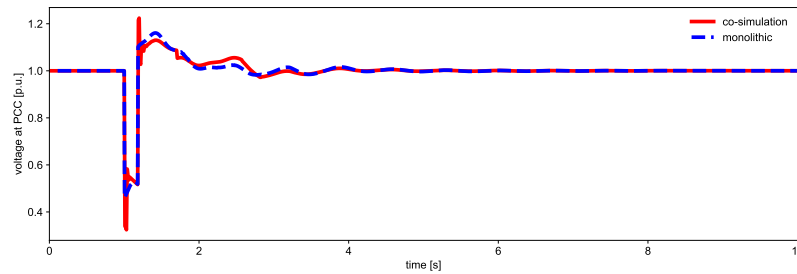


Fig. 3.7 Voltage magnitude at PCC, monolithic (red, solid) vs co-simulation (dashed, blue)

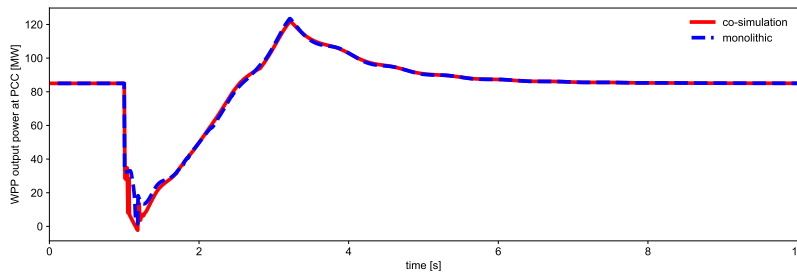


Fig. 3.8 WPP output active power, monolithic (red, solid) vs co-simulation (dashed, blue)

3.6 Conclusion

Major advancements in power electronic technology lead to its availability at all voltage levels and a massive deployment of components and systems grid-interfaced

though power electronics. This incredibly boosts the controllability of the system as a whole but also introduces coupling of phenomena in the time domain that could normally be addressed separately such as power system stability. Likewise, the digital transformation drastically increased the heterogeneity of the electricity system, transforming it from a purely physical system, showing continuous behaviour, to a cyber-physical system also exhibiting discrete-event behaviour (non-linear, discontinuous).

Simulation based assessment is a crucial link in the testing and validation chain of such integrated and intelligent energy systems (i.e. analysis, simulation, demonstration, roll-out). The heterogeneity of the (sub-)system models, however, shall be captured in the associated simulation tools accordingly. This is challenging as most simulators have been numerically optimised for a well-bounded domain, sometimes over decades. A solution for this challenge has been detailed in this chapter: co-simulation.

Various aspects of (coupling) simulation have been discussed: an overall typology for simulation-based assessment of CPES, the basics of co-simulation, standardised master algorithms and interfaces, and the framework approach adopted in the ERI-Grid project. More specifically, the ERI-Grid project achieved the following additions to the state-of-art in terms of co-simulation

- Readiness of the mosaik co-simulation framework for mutually coupled subsystems in the time-domain (i.e., cyclic dependencies);
- Implementation of an FMI++ adapter in PowerFactory (RMS mode) complying with the FMI for co-simulation specification. An application example has been discussed in Section 3.5;
- Development of an FMI++ export package for ns-3 based on FMI for co-simulation;
- Implementation of the FMI++ MATLAB toolbox based on FMI for co-simulation; and
- Proof-of-concept of continuous-time, discrete-event, and mixed simulator coupling;
- Assessment of the scalability of the applied approaches; and
- Application of the holistic testing methodology for simulation-based assessment methods.

Notwithstanding these innovations in terms of applications of co-simulations, the approach is not very suitable for the day-to-day engineer yet. Parameterisation of simulator interfaces, master algorithm configuration, distributed execution, and harmonisation of semantics of the overall simulation are examples that require a lot of manual work and are still subject to technological development. Once this is mature, the benefits are unprecedented: simulation-based assessment of heterogeneous CPES as a service.

References

1. Ieee standard for modeling and simulation (m & s) high level architecture (hla)– object model template (omt) specification (2010)
2. Barenblatt, G.I.: *Scaling*. Cambridge University Press (2003)
3. Bhandia, R., van der Meer, A.A., Widl, E., Strasser, T.I., et al.: D-JRA2.3 Smart Grid Simulation Environment. Deliverable D8.3, ERIGrid Consortium (2018)
4. Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., et al.: Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: *Proceedings of the 9th International MODELICA Conference*; September 3-5; 2012; Munich; Germany, 076, pp. 173–184. Linköping University Electronic Press (2012)
5. Garro, A., Falcone, A.: On the integration of hla and fmi for supporting interoperability and reusability in distributed simulation. In: *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 9–16. Society for Computer Simulation International (2015)
6. van der Meer, A.A., Bhandia, R., Widl, E., Heussen, K., Steinbrink, C., Chodura, P., Strasser, T.I., Palensky, P.: Towards scalable fmi-based co-simulation of wind energy systems using powerfactory. In: *Proceedings of Innovative Smart Grid Technologies (ISGT) Europe*. Bucharest, Romania (2019)
7. Palensky, P., van der Meer, A.A., López, C.D., Jozeph, A., Pan, K.: Applied co-simulation of intelligent power systems: implementation, usage, examples. *IEEE Industrial Electronics Magazine* **11**(2) (Jun 2017)
8. Rohjans, S., Lehnhoff, S., Schütte, S., Scherfke, S., Hussain, S.: mosaik - a modular platform for the evaluation of agent-based smart grid control. In: *IEEE PES ISGT Europe 2013*, pp. 1–5 (2013)
9. Steinbrink, C., van der Meer, A.A., Cvetkovic, M., Babazadeh, D., Rohjans, S., Palensky, P., Lehnhoff, S.: Smart grid co-simulation with mosaik and hla: a comparison study. *Computer Science - Research and Development* **23** (2017)