SMART GRID CO-SIMULATION BY DEVELOPING AN FMI-COMPLIANT INTERFACE FOR PSCAD

P. H. Divshali
VTT Research Centre of Finland
Poria.Divshali@vtt.fi

M. Laukkanen
VTT Research Centre of Finland
Matti.Laukkanen@vtt.fi

R. Bhandia
TU Delft, Netherland
R.Bhandia@tudelft.nl

A. A. VanderMeer
TU Delft, Netherland
A.A.vanderMeer@tudelft.nl

E. Widl
AIT Institute of Technology, Austria
Edmund.Widl@ait.ac.at

C. Steinbrink
OFFIS, Germany
Cornelius.Steinbrink@offis.de

A. Kulmala
VTT Research Centre of Finland
Anna.Kulmala@vtt.fi

K. Mäki
VTT Research Centre of Finland
Kari.Maki@vtt.fi

ABSTRACT

For smart grid assessment one needs to simulate varieties of components in different software environments. However, the existing simulation tools are domain oriented and cannot fulfil this need natively. Therefore, a smart grid simulation environment has to be established with accordingly accurate models for intra and inter-domain elements as well as interfaces and framework for coordination of those models in a holistic scenario. This paper presents part of the development of this smart grid simulation environment by implementing co-simulation interface for PSCAD using the mosaik framework based on functional mock-up interface (FMI). This co-simulation interface is tested using a modified dynamic model of IEEE 9 bus test system simulated in PSCAD while the wind turbine controller is simulated in MATLAB/Simulink. The results show a significant advantage over alternative methods in terms of a reduction in simulation runtime and compatibility with different simulation environments.

INTRODUCTION

A holistic model of smart grids includes several domains with fundamentally different nature, such as electrical systems, communication systems, control and prediction units, market models, and external factors like weather and people behaviour. From a conceptual point of view, this means that a smart grid system should be considered as a system-of-systems.

Each of these domains has a very different nature (continuous, discrete, stochastic, etc.) and they are using different modelling assumptions and implemented in different software packages. Therefore, there is a serious challenge in the holistic assessment of such system-of-systems by simulation studies. This is especially relevant to smart distribution grids, in which the behaviour of physical components is often dictated by an overlay system of ICT and distributed controls.

From the technical perspective, this challenge can be overcome using 1) multi-domain simulation environments or 2) co-simulation tools. There are several native multi-domain simulation environments, such as SystemModeler, Matlab/Simulink, Openmodelica. In addition, alternatively, a multi-domain simulation could be built from scratch. However, a simulator which supports multi-domain is not trivial to achieve due to the significant effort and expertise required. On the one hand, it is very often that the models from different domains would involve different environments and operating systems (i.e. 32 or 64 bit, windows, Unix or Linux). On the other hand, models from different domains often need to be dealt with using a different time scale, a model of computation (MoC) and specialized solvers. Building a simulator capable of providing appropriate environments, correct MoC, solvers and properly coordinating them internally is expensive and maybe not worth the effort [1], [2].

A further argument against a move to native multi-domain simulation environments is the need for models of professional quality in testing and operations assessments: domain experts are required to verify the correctness of models, which is easier to realize within domain-specific, accepted and validated tools.

In addition, from the modelling point of view, it is important to integrate commercial and open-source modelling and simulation frameworks, both specialized on particular system aspects (e.g. power system simulator or communication system simulator) and universal (e.g. general modelling environments like MATLAB/Simulink or Modelica-based tools). The specialized tools are usually equipped with validated component libraries, sophisticated import/export capabilities and well-designed user interfaces. The universal tools, on the other hand, are good for rapid prototyping of new and uncommon components, have extensive mathematical capabilities and are well-accepted in the scientific community. It is necessary to combine the best of both worlds.

In this regards, co-simulation techniques that provide a more powerful test environment are required. Using co-simulation techniques, the most suitable simulation tools for all considered domains, from the perspective of accuracy and runtime, can be coupled [3].

For this purpose, the ERIGrid project [4], a multi-partner project funded by EU-Horizon 2020, aims to develop and validate a holistic model for smart grids. The co-simulation activities in ERIGrid focus on developing co-simulation tools using the mosaik framework [5] based on the emerging industry standard functional mock-up interface (FMI) [6].

This paper focuses on the architecture and the development of a co-simulation interface for PSCAD as a part of this holistic simulation tools for smart grids. The PSCAD co-simulation interface is tested by modelling the controller of an onshore wind power plant implemented in MATLAB/Simulink while it is interconnected to a modified dynamic model of IEEE 9 bus test system implemented in PSCAD.

CO-SIMULATION CHALLENGES

In order to assess a system-of-systems using co-simulation, it is necessary to integrate the MoC behind a model or a simulator. The MoC represents the interactions between modules, components or phenomena and it is independent of the implementation technology (i.e. sequential or parallel) and language (i.e. Matlab, Python). MOC can be classified as: Imperative (e.g. Emulators), Finite State Machine (e.g. a set of states, rule-based control), Dataflow (e.g. ODEs, DEAs), Discrete Event (e.g. communication, zero-crossing), etc. [7].

The energy domain simulators often employ Dataflow MoC due to the fact that they derive mostly from sets of ordinary differential equations defining the state variables and the environmental factors of a system (e.g. steady-state simulations, electromagnetic transients or circuit simulations). However, ICT, market simulator and eventually control simulators use often the Discrete Event or Finite State Machine MoC.

The discrete models react to events that occur at a given time instant and produce other events either at the same time instant or at some future time instant in a chronological execution order. Combining discrete event and continuous simulation requires mixing different MoC such as Discrete Events and Dataflow in a hierarchical way [8]. It leads to the necessity of an interaction semantic that resolves the ambiguities caused by differences among MoC. Events that cross the domains need to be totally ordered and associated with timestamps. Moreover, each domain (simulator and MoC) must also support a rudimentary notion of time. The main difficulties for integration of different MoC involve how to deal with simultaneous events and zero-delay feedback loops [7].

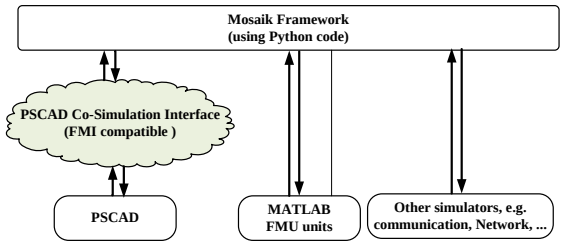

Fig. 1: The proposed PSCAD co-simulation structure.

In these circumstances, the co-simulation platform must overcome the following challenges:

- Modularity, hierarchical composition, and proper linking of domain-specific tools to maintain the synchronism between the simulators while the cyclic dependencies are implemented between the co-simulation interfaces.
- Scenario handling and system handling to allow easy component replacement.
- Distributed and parallel simulation.

PROPOSED APPROACH

In order to overcome the abovementioned challenges, the co-simulation interface has been developed using the mosaik framework based on the FMI interface. Fig. 1 shows the structure of the co-simulation method used in this project.

FMI Interface

The FMI interface provides the most essential and fundamental functionality to handle and manipulate functional mock-up units (FMU), such as numerical integration, advanced event-handling or state predictions.

However, the FMI is in the form of a C interface and implies several prerequisites that a simulation tool has to fulfil in order to be able to utilize such an FMI component. Therefore, the *FMI++ Python Interface* has been developed, a Python package wrapping the *FMI++ Library* [9]. For this purpose, the ERIGrid project combines the FMI++ python interface and mosaik framework for co-simulation of FMU.

Mosaik Framework

As mentioned previously, the holistic smart grid simulator requires means of flexible scenario handling in complex setups. Such feature allows for easy component replacement and system scaling, which makes up the main advantages simulation-based validation has over lab-based validation. For this reason, the mosaik platform is used.

The mosaik 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

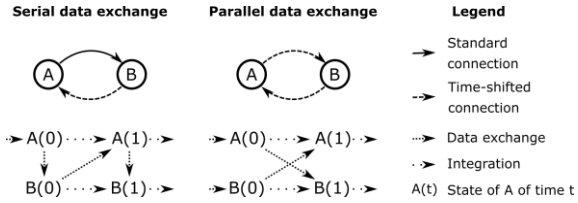

Fig. 2: Data exchange schemes possible in mosaik.

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 [5].

The co-simulation framework mosaik has been developed with a strong focus on flexibility. However, in order to guarantee the absence of deadlocks for any given setup, the handling of so-called cyclic dependencies in mosaik has so far had some limiting characteristics. In particular, using mosaik's intuitive connection capabilities to establish a 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 little programming experience.

Within the scope of the ERIGrid project, the capabilities of mosaik have been extended to allow for higher usability in the handling of cyclic dependencies. The basic idea of the extension is the separation of data exchange into two stages: simulators may either receive data *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 limited.

Fig. 2 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. As Fig. 2 shows, 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. This way, parallel data exchange schemes may also be realized if initial input data can be assigned to each simulator.

PSCAD CO-SIMULATION

As a part of the holistic simulation tools developed for

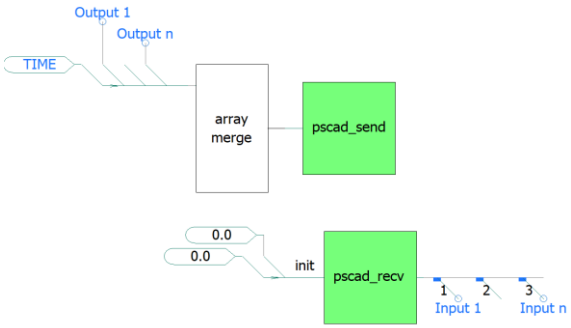

Fig. 3: the PSCAD co-simulation interface

smart grids in the ERIGrid project, an FMI-compliant interface is developed for PSCAD. PSCAD is a commercial general-purpose time domain simulation tool for studying transient behaviour of electrical networks. Its strengths, in addition to its computational performance and advanced user interface, are in the modelling modularity and ability to model the components using standard library components or user-built model components of the desired level of detail. The most recent versions of PSCAD provide a built-in interface for MATLAB/Simulink. However, this interface proved to perform poorly in practice, as data exchange was remarkably slow.

PSCAD provides so-called automation API, a Python interface which can be used to interact with the software, e.g. opening and loading projects, setting model parameters and running simulations. Despite providing a wide range of functions, the API cannot access the simulation model's data signals that are needed for co-simulation. However, user components in PSCAD are programmable with Fortran, which in turn allows cross-compiling C/C++ code and thus enables developing co-simulation interface. Using this feature, this project developed the PSCAD co-simulation interfaced, shown as the green box in Fig. 1, to add FMI ability to PSCAD. This co-simulation interface is implemented using a C-function to read and write into TCP/IP protocol and is merged with PSCAD using Fortran compiler.

This co-simulation interface consists of two parts: A back-end component used by PSCAD, and a front-end component used by the master algorithm. The back-end part is implemented as user-defined components called pscad_send and pscad_recv (See Fig. 3). They take desired input/output signal names as arrays, and at configurable time intervals (co-simulation time-step) they send/receive data over a socket. Socket communication functions are written in C.

The front-end component was initially implemented as a kind of FMU, a Python program that acts as a socket server, exchanging data with PSCAD and exposing it to the master algorithm. It also handles the time

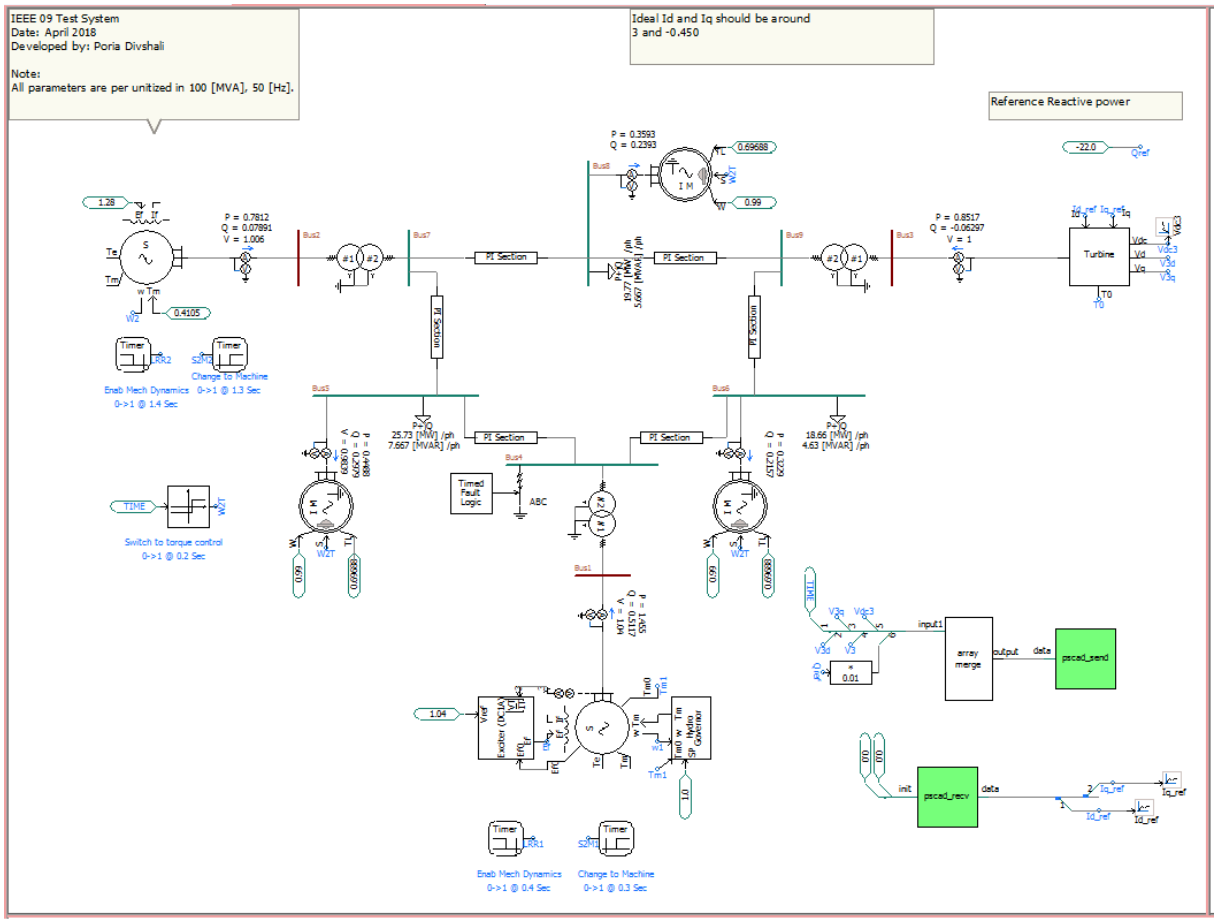

Fig. 4: The dynamic model of IEEE 9 bus system, implemented in PSCAD

synchronization between the master and the slave. The developed co-simulation interface, which is compatible with the FMI standard can be found in ERIGrid GitHub [10]. However, further work is required to add the ability of encapsulating the PSCAD simulation and exporting it as an FMU.

VALIDATION

In order to validate the PSCAD co-simulation package, a modified dynamic model of IEEE 9 bus test system [11], replacing the third generator by a wind turbine (WT), is implemented in PSCAD. In monolithic simulation case, the WT controller is also implemented in PSCAD but in co-simulation case, the control mechanisms of WT are implemented in MATLAB/Simulink and exported as an FMU.

The dynamic model of the modified IEEE 9 bus system is implemented in PSCAD as shown in Fig. 4, while a three phase fault happened at $t = 0.4$ s. The third synchronous generator, G3, is replaced by a WT at the modified IEEE 9 bus test system. This WT is modelled by a converter, Turbine block in Fig. 4, in PSCAD.
The PSCAD simulation sends and receives signals using the pscad_send and pscad_recv blocks, developed for co-simulation interface of PSCAD. By using this interface, the co-simulation can be conducted through FMI++ interface. These blocks send some measurements, which are required for controllers implemented in SIMULINK, e.g. time, *Vd*, and *Vq* and receive the output of the controllers, the reference values for the converter, via the master program.

In order to validate the co-simulation, the results of the co-simulation are compared with that from the results of PSCAD monolithic simulation. Fig. 5 shows this comparison. The simulation is performed for 20 seconds; since PSCAD starts from zero initial condition, the first 10 seconds (before 0) is used for the model initialising.

As shown in Fig. 5, the co-simulation and monolithic simulations trace the same and overlap each other indicating that the co-simulation produces the same results as monolithic. It is noteworthy to mention, there was not considerable difference between simulation runtime of monolithic and co-simulation cases. The implementation of both monolithic and co-simulation cases are available in [10].

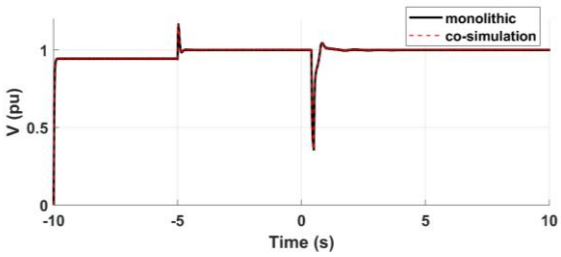

Fig. 5.a: The output voltage of the WT

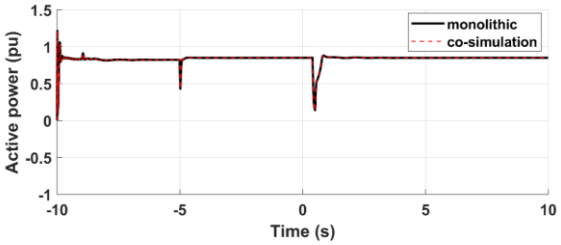

Fig. 5.b: The active power output of the WT

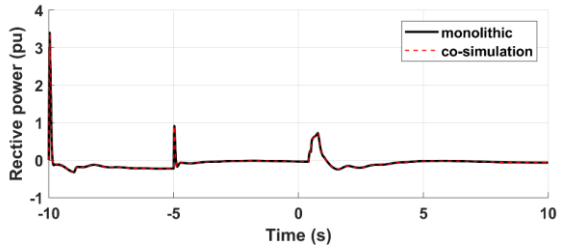

Fig. 5.c: The reactive power output of the WT

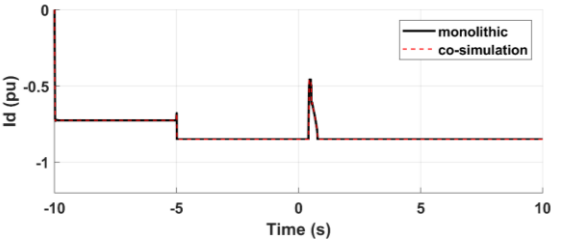

Fig. 5.d: The reference current (Id) of the WT

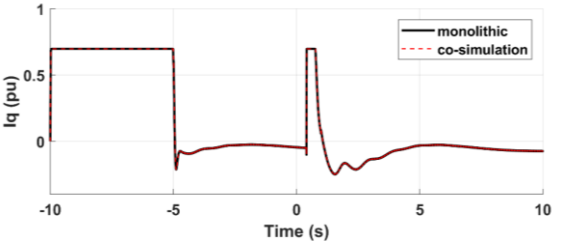

Fig. 5.e: The reference current (Iq) of the WT

CONCLUSION

This paper presents the challenges of developing a holistic simulation tool for assessing smart grids as a system of systems. Since the smart grids include different domains a co-simulation interface coupling power system to other domains, such as ICT and control, is necessary. This paper develops a co-simulation interfaced for PSCAD using the mosaik framework based on FMI. The results validate and demonstrate the effectiveness of the proposed method. It is worth to mention that although the validation case performed by co-simulation of PSCAD and Matlab, the developed interface enables interfacing also with other programs.

REFERENCES

[1] K. Johnstone, S. M. Blair, M. H. Syed, A. Emhemed, G. M. Burt, and T. I. Strasser, "Co-simulation approach using PowerFactory and MATLAB/Simulink to enable validation of distributed control concepts within future power systems," CIRED - Open Access Proc. J., vol. 2017, no. 1, pp. 2192–2196, Oct. 2017.

[2] A. Latif, M. Shahzad, P. Palensky, and W. Gawlik, "An alternate PowerFactory Matlab coupling approach," in Proceedings - 2015 International Symposium on Smart Electric Distribution Systems and Technologies, EDST 2015, 2015.

[3] T. Strasser et al., "Towards holistic power distribution system validation and testing—an overview and discussion of different possibilities," Elektrotechnik und Informationstechnik, 2017.

[4] "ERIGrid Project." [Online]. Available: https://erigrid.eu/. [Accessed: 12-Sep-2018].

[5] "mosaik — A flexible Smart Grid co-simulation framework." [Online]. Available: https://mosaik.offis.de/. [Accessed: 12-Sep-2018].

[6] "Functional Mock-up Interface." [Online]. Available: https://fmi-standard.org/. [Accessed: 12-Sep-2018].

[7] Szu-Chu Liu and Shyang Chang, "Dimension estimation of discrete-time fractional Brownian motion with applications to image texture classification," IEEE Trans. Image Process., vol. 6, no. 8, pp. 1176–1184, 1997.

[8] J. Banks, Discrete-event system simulation. Prentice Hall, 2010.

[9] "The FMI++ Python Interface for Windows." [Online]. Available: https://pythonhosted.org/fmipp/. [Accessed: 13-Dec-2018].

[10] "ERIGrid JRA2: Test case TC1 mosaik implementation." [Online]. Available: https://github.com/ERIGrid/JRA2-TC1. [Accessed: 31-Dec-2018].

[11] P. Demetriou, M. Asprou, J. Quiros-Tortos, and E. Kyriakides, "Dynamic IEEE Test Systems for Transient Analysis," IEEE Syst. J., 2015.