# A Variable-Rate Co-Simulation Environment for the Dynamic Analysis of Multi-Area Power Systems

Claudio David López, Arjen A. van der Meer, Miloš Cvetković and Peter Palensky
Department of Electrical Sustainable Energy
Delft University of Technology
The Netherlands

***Abstract*—The unprecedented complexity of modern power systems has created a need for analyzing the interactions between different power system areas, which requires detailed physical models of all involved grids. However, a single institution seldom has access to enough information to build a complete model of a multi-area system. Additionally, such a model would be too labor-intensive to build and too computationally expensive to simulate. Co-simulation is an alternative that allows different institutions (TSOs, DSOs, research institutes, etc.) to simulate cooperatively by interconnecting their simulation tools, without having to disclose their grid models, and while sharing both the burden of model development and the computational load of the co-simulation. We present a co-simulation environment designed for researching the variable-rate (variable time step size) synchronization methods needed in a multi-institution setting. The environment can couple an arbitrary number of instances of DIgSILENT PowerFactory running on different virtual servers, at different rates, each representing a different area. An example use case with a three area system illustrates some of the main features of this environment. Errors bellow 5 % are evidence that this type of co-simulation is feasible, but long execution times point to additional challenges.**

***Index Terms*—Co-simulation, distributed simulation, master algorithm, multi-area, PowerFactory, transmission systems**

## I. Introduction

Modern power systems have reached an unprecedented level of complexity due to the introduction of renewable energy generation, energy storage, power electronics and advanced control strategies. The widespread and distributed presence of these technologies has made the interaction between different power system areas harder to understand and predict. As a consequence, simulations that allow the study of interactions between interconnected areas are increasingly necessary.

Traditional power system simulations rely on a detailed model of the area of interest and simplified equivalent models of all neighboring areas. Yet, simplified models cannot ensure that the interactions between areas are captured in sufficient detail; analyzing these interactions demands detailed physical models of all involved grids. This is challenging because detailed information about all areas in a system is seldom available to a single institution, because building and maintaining models of such proportions is extremely labor intensive, and because simulating them would be computationally prohibitive. Such a situation occurs, for example, when multiple TSOs must carry out integration studies of their grids.

Co-simulation is an alternative approach to this simulation problem. Following this approach, each area can be modeled and simulated in a different simulation tool (or simulator), and interconnections between areas can be established by interconnecting corresponding simulators. A *co-simulation master* is in charge of orchestrating a synchronized data exchange between simulators [1]. This approach has several advantages:

- Each grid model can be developed and maintained independently by the institution that has access to the required information.
- Each institution can continue to develop models for and run simulations with the tool of its choice, since information must be exchanged between simulators through a standard, tool-independent protocol.
- Institutions do not need to share or disclose their models, or any private information about them, since co-simulation only requires selected variables to be exchanged at run time.
- The burden of model development and maintenance can be shared among institutions.
- The computational load of the co-simulation can be shared among institutions, since each simulator can run on a different computer if data is exchanged over a network.

However, co-simulations often produce less accurate results than monolithic simulations [2]. Additionally, the potentially large number of simulators required to co-simulate an entire power system and the possibility that some of them adjust their time step at run time considerably increases the complexity of simulator synchronization, which in turn can increase execution time. Moreover, it becomes necessary to initialize independent simulators through power flow calculations that are mutually consistent.

The accuracy and numerical stability of co-simulations have been widely researched for the case of two dynamic simulators (e.g., [2]). Typically, both simulators use the same time step size, or the step size of one is a multiple of the other. There is still a need for research on how different synchronization methods influence accuracy for more than two simulators, where at least one runs at a variable rate (variable step size).

Co-simulations that couple an arbitrary number of simulators have mostly been researched from the software viewpoint.

Notable examples are environments based on mosaik (e.g., [3]) and on the high-level architecture or HLA (e.g., [4]). Nevertheless, these environments are currently inadequate for researching variable-rate, dynamic co-simulations of multi-area systems. In the case of mosaik, implementing the bidirectional data exchange needed in dynamic co-simulations is cumbersome, and only one synchronization method is available [5]. In the case of the HLA, coupling continuous simulators is not straightforward, as the HLA was originally developed for event-based simulators [4]; this can be especially troublesome if interpolation and/or extrapolation of simulator outputs is needed, which is the case of dynamic, variable-rate co-simulations. Furthermore, both mosaik and the HLA impose requirements that most traditional dynamic power system simulators struggle to comply with. Thus, a tool that enables research on how to best couple and synchronize these simulators is needed.

To address this need, our paper presents the first step in the development of a co-simulation environment for researching the effect of different coupling and synchronization methods on the accuracy and computational performance of variable-rate, dynamic, multi-area power system co-simulations. The environment is composed of a set of instances of DIgSILENT PowerFactory running on different virtual servers that represent the simulators of different institutions (TSO, DSO, research institute, etc.), and a co-simulation master that provides facilities that allow the user to implement different synchronization methods. The environment imposes very simple requirements on the simulators, making it is easy to include simulators other than PowerFactory.

The paper is structured as follows: Section II summarizes the design and implementation of the environment, Section III focuses on the initialization and synchronization of simulators, Section IV introduces an example use case, and Section V presents the conclusions.

## II. Design and Implementation of the Environment

The co-simulation environment consists of a set of PowerFactory simulators and a co-simulation master, as illustrated in Fig. 1. Each instance of PowerFactory runs on a different virtual server. The only requirement the environment imposes on the simulators is that they are able to send their outputs to the master through a TCP/IP socket connection and wait for inputs from the master before executing a new time step. The master provides facilities for implementing user-defined synchronization methods. Since PowerFactory requires the use Windows servers, the environment is managed through PowerShell remote sessions.

### A. Simulators

PowerFactory is a simulator capable of running dynamic simulations of transient stability type (TS), also referred to as root mean square (RMS), and of the electromagnetic transient type (EMT). Unless the real time mode is used, this simulator uses adaptive time step solvers that adjust the simulation step when an event occurs (e.g., short circuit, load, tap change events). To send and receive data, each simulator must implement a DSL[1] block that reads data from the grid model and sends it through a socket connection, and receives data from the socket connection and sets it inside the grid model. This is done each time step using the external DLL[2] from [6].

Using the Python API provided by PowerFactory, a script was developed that starts PowerFactory in engine mode, configures it, runs the simulation, and retrieves the results of interest. This script requires the following information to be specified as a command line argument in a JSON[3]-formatted object:

- IP address of the server where the master resides.
- TCP port where the master expects the simulator to connect.
- Simulation type (RMS or EMT).
- Start time, end time and (maximum) time step size.
- Instructions for result monitoring and retrieval.

### B. Co-Simulation Master

The co-simulation master is composed of a set of *simulator proxies*, which are simulator abstractions and act as the interface between the master and the simulators, a *synchronization module* that enables coordinated data exchange between proxies, and a *synchronization request queue* that serves as a signaling mechanism to indicate that synchronization between simulators is needed. This structure is shown in Fig. 1.

The co-simulation master was implemented as a Python script that expects the following information to be provided as a command line argument in a JSON-formatted object:

- IP address of the server where the master resides.
- TCP port where the master expects each simulator to connect.
- Unique name for each simulator.
- Number of outputs of each simulator.
- Source of the inputs of each simulator.

*1) Simulator Proxy Modules:* A simulator proxy module is as an abstraction of a simulator designed to simplify the interaction between the co-simulation master and a simulator. Simulator proxies are in charge of handling the socket connections through which data is exchanged, and of providing a standard interface that the synchronization module can use to enable coordinated data exchange between simulators.

To handle the socket connection, the simulator proxy must be capable of accepting a connection, converting data between the binary formats used within the TCP/IP channel and within the co-simulation master, sending and receiving data, and safely closing the connection once it is no longer needed. Since a simulator can send data to the master at any moment, a dedicated thread is in charge of receiving data from the simulator asynchronously.

[1] DIgSILENT Simulation Language
[2] Dynamic-link library
[3] JavaScript Object Notation

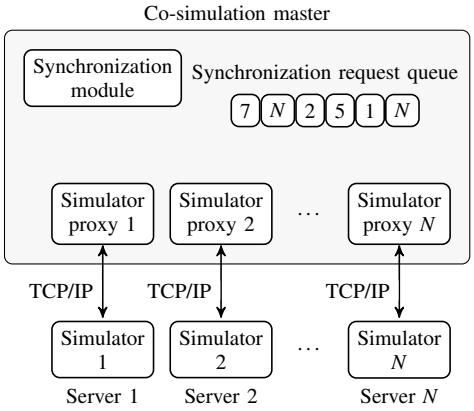


Fig. 1. Structure of the co-simulation environment.

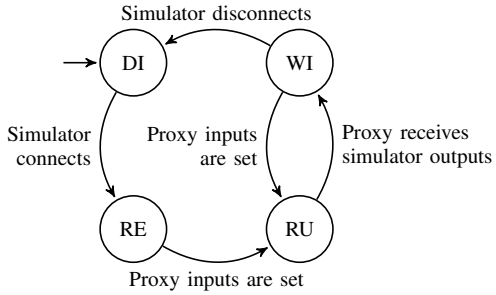


Fig. 2. State diagram of a simulator proxy (DI: disconnected, RE: ready, RU: running, WI: waiting for inputs).

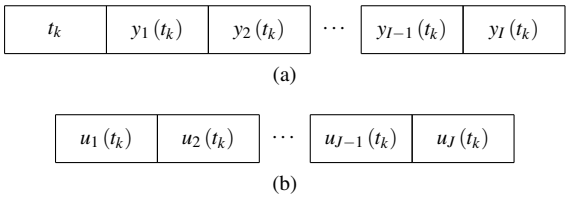


Fig. 3. Structure of a: (a) simulator output message of length $I$ values plus time stamp, (b) simulator input message of length $J$ values. Each value is double-precision floating-point (8 bytes).

These modules are implemented as finite state machines that can be in one of four states (see Fig. 2): The *disconnected* state indicates that no socket connection between the proxy and the simulator has been established, *ready* that a connection has been established and the simulator is ready to start a simulation, *running* that the simulator is executing a time step, and *waiting for inputs* that the simulator has just finished a time step, has sent its outputs to the co-simulation master and is awaiting inputs.

When data arrives, the simulator proxy first appends it to a fixed-size FIFO buffer from which it can be retrieved in the future (if required) and then signals a need for synchronization by placing a request in the synchronization request queue. Because not all simulators can be expected to run at the same rate, a given simulator may need the outputs of another simulator at a time for which outputs are not produced. In this case, the values of these outputs at the required simulation time can be estimated by interpolating the outputs already available in the aforementioned FIFO buffer.

The interface that each simulator proxy provides to the synchronization module is implemented through a set of four methods that

- retrieve simulator outputs at a given simulation time,
- retrieve the current simulator time,
- retrieve the state of the simulator proxy,
- and set the inputs of a simulator.

*2) Synchronization Module:* The synchronization module is in charge of exchanging data between source and destination simulators, and is the only module in the co-simulation master that needs to be aware of the topology of the co-simulation. The synchronization module is automatically executed when there is a synchronization request in the queue. When executed, this module must determine how to proceed based on the simulator time, the state of each proxy, and a user-defined synchronization method. If the inputs that a (destination) proxy requests can already be retrieved from the remaining (source) proxies, the synchronization module can retrieve them from the source proxies and set them in the destination proxy. Once the inputs are set, the simulator connected to the destination proxy can resume execution.

*3) Synchronization Request Queue:* The synchronization request queue is an asynchronous signaling mechanism that simulator proxies use to indicate a need for synchronization to the synchronization module (need for inputs from other simulators). Each request contains an identifier of the proxy that placed it. The purpose of this queue is to ensure that no synchronization request goes unattended, even if the synchronization module is busy when the request is placed.

### C. Simulator Input and Output Messages

All messages that are received in and sent from a simulator contain double-precision floating-point values only. Simulator output messages are always time-stamped for synchronization purposes, while input messages are not (see Fig. 3). The messages may contain any numerical values that need to be exchanged.

### D. Configuration

To configure the environment, the information that each simulator and the master expect as command line arguments must be specified. This is done in a single JSON file that must contain the IP address of the server where the co-simulation master is located, as well as the following information for each individual simulator:

- Unique simulator name.
- TCP port where the master expects the simulator to connect.
- Number of simulator outputs.
- Source of each simulator input.

The source of each input is specified as a list where each entry is a combination of source simulator name and output

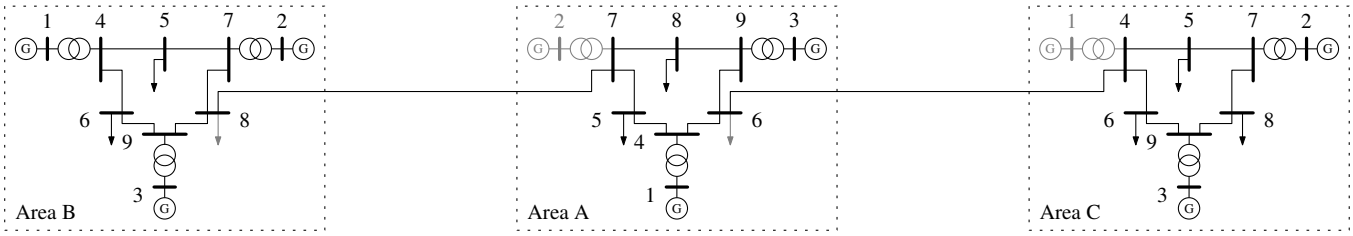


Fig. 4. Co-simulated three area system. Each area is a modified IEEE 9 bus system as specified in [7], and is simulated in a different instance of PowerFactory running on a different virtual server. Out of service elements are colored gray.

number. For example, if the inputs of a given simulator are specified as `[["B", 0], ["B", 3], ["C", 0]]` in the JSON configuration file, it means that input 0 comes from output 0 of simulator B, input 1 comes from output 3 of simulator B, and input 2 comes from output 0 of simulator C.

### E. Management of the Environment

All processes that take part in the co-simulation environment (i.e., simulators and co-simulation master) run on different servers. This means that set-up, start-up and result retrieval must be done remotely. Since the Python scripts that perform these actions are configured through command line arguments, these actions can be performed by establishing PowerShell remote sessions with each virtual server. This can be automated in a PowerShell script.

## III. Initialization and Synchronization of Simulators

Each simulator in the co-simulation needs initial conditions that are consistent with those of the remaining simulators. At the same time, once a common starting point for all simulators is found, data exchange between simulators must be coordinated with the simulation time of each one of them.

### A. Initialization Method

PowerFactory determines initial conditions through power flow calculations. To determine a set of initial conditions that is consistent throughout the entire co-simulation, an equilibrium point must be found for every interface between simulators. This equilibrium exists when voltage and current are the same on each side of each interface after a power flow calculation is executed on each individual simulator. The equilibrium can be determined using an iterative initialization method. The exact implementation of such an iterative method depends on the way the interfaces between simulators are defined, but the general idea remains the same. For example, if the interfaces use the ideal transformer method [8], which consists of an ideal voltage source on one side of the interface and an ideal current source on the other, the iterative initialization method can be implemented as follows:

1) Execute a power flow calculation on each simulator.
2) Measure the voltage at the bus where the current source is connected.
3) Measure the current flowing through the voltage source.
4) Set the voltage from step 2 in the voltage source.
5) Set the current from step 3 in the current source.
6) If the voltage and current on each side of the interface differ from those of the previous iteration, repeat from step 1. Otherwise, stop iterating.

This procedure must be carried out simultaneously for every interface in the co-simulation.

### B. Synchronization Method

To illustrate how a variable-rate synchronization method can be implemented using the four methods provided by simulator proxies (see Section II-B1), let us define a simple synchronization method that always sets the inputs of the simulator lagging the furthest in time. If all other simulators are ahead, this means that the inputs that the lagging simulator requires to continue execution are either already available or can be interpolated from the available ones. This synchronization method can be implemented as follows:

1) When attending a new synchronization request, wait until all proxies are in the waiting for inputs state.
2) Determine which simulator is lagging the furthest in simulation time.
3) Get the outputs of the simulators that the lagging simulator depends on.
4) Set the inputs of the lagging simulator and go back to step 1.

## IV. Example Use Case: Co-Simulation of a Three Area System

As an example, let us consider a three area system where each area is simulated in a different simulator and a fault in one area makes one of the simulators dynamically adjust its time step. A monolithic simulation of the same system is carried out as well as a benchmark.

### A. Co-Simulation Scenario

The co-simulated system is composed of three areas (A, B and C). Each area is a modified IEEE 9 bus system as specified in [7]. Areas are interconnected by tie lines that connect a generator bus and a load bus (see Fig. 4). The loads and generators originally connected to these buses are placed out of service to enable the flow of power between areas. A three-phase short circuit occurs at bus 4 in area C at 0.1 s.

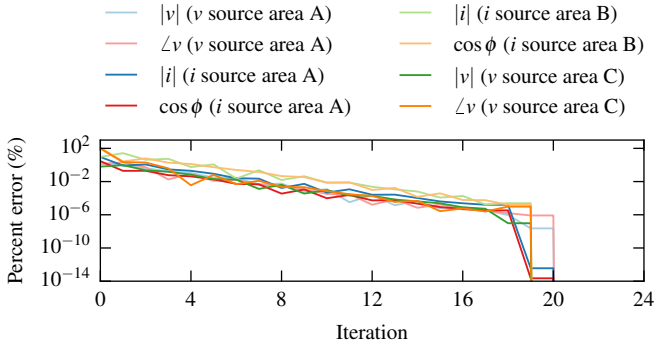


Fig. 5. Evolution of the percent error of every interface variable (voltage magnitude and angle for every voltage source, current magnitude and power factor for every current source) during initialization. The percent error is calculated with respect to the equilibrium values. The vertical axis is in logarithmic scale.

### B. Co-Simulation Implementation

The simulators are set in EMT mode with a maximum time step of 20 µs (dynamically adjusted in area C after the short circuit). The interfaces between simulators are implemented with the ideal transformer method [8]. All interface voltage sources are initially set to $1\angle 0°$ p.u., whereas current sources are set to 100 A and unity power factor. The initialization and synchronization methods described in Section III are used.

### C. Environment Implementation

The co-simulation environment is implemented with four virtual servers, each running Windows Server 2012 R2 (64 bits) on a single core with 2 GB of RAM, Python 3.4, PowerFactory 15.2, and PowerShell 5.0. The co-simulation master runs on one server, while the remaining three servers are used for the simulators.

### D. Results and Discussion

Fig. 5 shows the percent error at each interface between simulators during initialization (note the logarithmic scale on the vertical axis). The error is calculated with respect to the equilibrium voltage and current values at each interface. Twenty iterations are needed until full convergence is achieved and the error is negligible. Nevertheless, after 10 iterations, all interfaces already show an error below 0.0075 %. When all interfaces are in equilibrium, the power flow results from the co-simulation match those from the monolithic simulation. Although several iterations are needed until full convergence is reached, once the right set points for each interface are found and recorded (voltage magnitude and angle for every voltage source, current magnitude and power factor for every current source), no iterations are needed for initializing further co-simulation runs if the same set points are reused.

The voltage at one bus per area and the current flowing between areas were selected to characterize the time-domain co-simulation results. Fig. 6 compares the voltage waveforms obtained from the co-simulation to those obtained from the monolithic simulation. Here, no difference can be observed

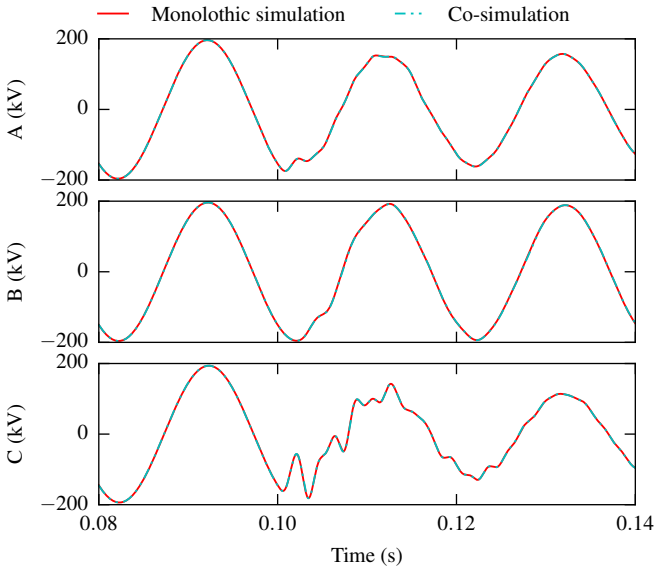


Fig. 6. Voltage at bus 9 (one phase) in areas A, B and C: comparison between a monolithic simulation (benchmark) and the co-simulation. A three-phase short circuit occurs at bus 4 in area C at 0.1 s. Monolithic simulation results overlap with co-simulation results.

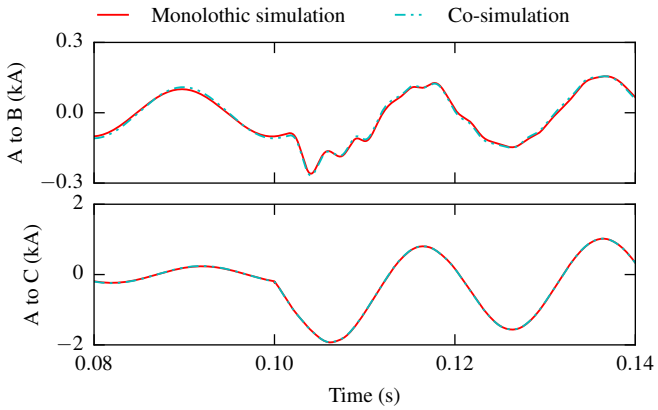


Fig. 7. Current (one phase) flowing between areas: comparison between a monolithic simulation (benchmark) and the co-simulation. A three-phase short circuit occurs at bus 4 in area C at 0.1 s. Monolithic simulation results overlap with co-simulation results in the case of the current between areas A and C.

between co-simulation and monolithic simulation, even after the three phase short circuit at 0.1 s. Fig. 7, which compares current waveforms, shows a marginally less favorable result. A more detailed comparison is made in Fig. 8, which shows the percent error between co-simulation and monolithic simulation results with respect to peak values. Here, the error remains below 1 % in the case of voltage and bellow 5 % in the case of current. Note how the error behaves differently after the short circuit at 0.1 s, when the simulator of area C starts adjusting its time step dynamically.

Although the co-simulation results are remarkably accurate when compared to those of the monolithic simulation, a maximum time step of 20 µs is required to achieve this accuracy, a rather small time step even for an EMT simulation. This time

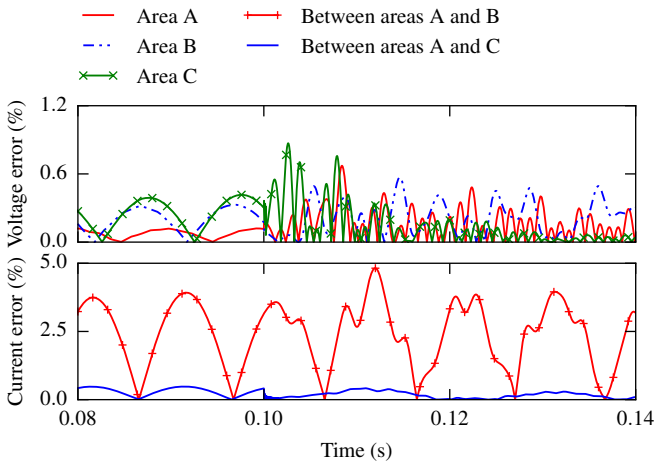


Fig. 8. Percent error between co-simulation and monolithic simulation results with respect to peak values for the voltage at bus 9 in every area and the current flowing between areas.

step size was chosen since the accuracy of the co-simulation results decreases rapidly as the time step size increases, and the co-simulation is already numerically unstable for a time step size of 50 µs. This is not surprising considering that the ideal transformer method has been shown to be accurate but prone to instability [8].

The co-simulation takes approximately 7 min to complete, which contrasts with the approximately 1.5 s that the monolithic simulation of the same system requires under the same conditions. There are several factors that must be taken into account to understand this difference and to identify measures that can improve computational performance:

- *Communication and synchronization overhead:* Between time steps, additional time is required for communication between the co-simulation master and each simulator, and for the execution of the co-simulation master.
- *Virtual server performance:* The virtual servers used are not high-performance. Better performing servers would reduce calculation time and synchronization overhead.
- *Implementation of the co-simulation master:* Using a scripting language like Python without taking performance optimization measures may lead to an immoderate synchronization overhead.
- *Time step size:* A small time step not only increases the total number of calculations each simulator executes, but also the total communication and synchronization overhead, since more data is exchanged between the co-simulation master and each simulator.
- *Interface method:* Using an interface method other than the ideal transformer method could allow larger time steps, thus decreasing execution time.
- *Synchronization method:* Serial (Gauss-Seidel) synchronization is used, which typically yields longer execution times than a parallel (Jacobi) method. However, serial methods are known to be slightly more accurate [1]. Additionally, synchronization is done with a *lazy* approach, where simulators only take a step when they must. A parallel synchronization method with a more proactive approach, in which simulators take a step whenever they can, could increase performance.
- *Size of models:* If the models are small, as is the case with the 9 bus system, the communication and synchronization overhead is significant compared to the time simulators require to solve the model equations. Even if the synchronization method implements some level of parallelization, the benefit might not outweigh the overhead. In the case of large models, the time required to solve the models can become dominant compared to the overhead and parallelization could offer a performance advantage with respect to a monolithic simulation.

## V. Conclusion

This paper introduced a PowerFactory-based co-simulation environment for researching synchronization methods for variable-rate, dynamic co-simulations of multi-area power systems, and presented an example use case with a three area system. The co-simulation results show errors below 5 %, even when one of the areas dynamically adjusts its time step. However, the execution time reached approximately 7 min, while a monolithic simulation of the same system required only 1.5 s to complete. Despite this drawback, the observed accuracy and the numerous measures to potentially reduce execution time indicate that dynamic co-simulation for multi-area power systems is feasible. With the help of the presented environment it becomes easier to explore these measures and develop methods to reap the full benefits of co-simulation for this particular application.

## References

[1] P. Palensky, A. A. van der Meer, C. D. López, A. Joseph, and K. Pan, "Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, Mar. 2017.

[2] M. Busch, *Zur effizienten Kopplung von Simulationsprogrammen*. Kassel University Press GmbH, 2012.

[3] M. Buscher, A. Claassen, M. Kube, S. Lehnhoff, K. Piech, S. Rohjans, S. Scherfke, C. Steinbrink, J. Velasquez, F. Tempez, and Y. Bouzid, "Integrated smart grid simulations for generic automation architectures with RT-LAB and mosaik," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Nov. 2014, pp. 194–199.

[4] M. U. Awais, W. Mueller, A. Elsheikh, P. Palensky, and E. Widl, "Using the HLA for distributed continuous simulations," in *2013 8th EUROSIM Congress on Modelling and Simulation (EUROSIM)*, Sep. 2013, pp. 544–549.

[5] S. Scherfke, *mosaik Documentation Version 2.3.0*, OFFIS.

[6] M. Stifter, F. Andrén, R. Schwalbe, and W. Tremmel, "Interfacing PowerFactory: Co-simulation, real-time simulation and controller hardware-in-the-loop applications," in *PowerFactory Applications for Power System Analysis*, F. M. Gonzalez-Longatt and J. L. Rueda, Eds. Springer International Publishing, 2014, pp. 343–366.

[7] L. Hadjidemetriou, A. Charalambous, P. Demetriou, and E. Kyriakides, "Dynamic modeling of IEEE test systems including renewable energy sources," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, Apr. 2016, pp. 1–6.

[8] W. Ren, M. Steurer, and T. Baldwin, "Improve the stability and the accuracy of power hardware-in-the-loop simulation by selecting appropriate interface algorithms," *IEEE Transactions on Industry Applications*, vol. 44, no. 4, pp. 1286–1294, Jul. 2008.