Evaluation of smart grid control strategies in co-simulation - integration of IPSYS and mosaik

Anna Magdalena Kosek* and Ontje Lünsdorf † Stefan Scherfke † Oliver Gehrke* Sebastian Rohjans †
* Technical University of Denmark, Department of Electrical Engineering
Energy Systems Operations and Management
Frederiksborgvej 399, Building 776, 4000 Roskilde, Denmark
Email: {amko,olge}@elektro.dtu.dk
† OFFIS – Institute for Information Technology
Escherweg 2, 26121 Oldenburg, Germany
Email: {ontje.luensdorf,stefan.scherfke,sebastian.rohjans}@offis.de

***Abstract*—This paper presents two different aspects considering a co-simulation of smart grid scenarios. First considers representing the control strategy in a separate discrete event simulation developed in a multi-agent platform. This study investigates the design and implementation of such a simulator. Special attention is given to timing issues presenting time variant and time invariant models. The second aspect presented in this paper is the co-simulation composition, investigating how to integrate a control simulation with other simulators in a co-simulation ecosystem. In this study the attention is given to the co-simulation scheduling, proposing two integration approaches: overall control and separate domain. Results from a proof-of-concept implementation are included.**

I. Introduction

The increasing share of renewable resources in the European power gr id brings new challenges. Consumption driven power system need to be transformed into production driven. One of the challenges tackled in Smart Grid is enabling controllability of Distributed Energy resources. Other challenges include for example power grid stability, including voltage and frequency control, and dealing with congestion. In order to face these challenges, grid operation has to become more flexible which requires a much higher penetration of ICT. This means that classical power system simulation models - focused on an electrical model of the system - falls increasingly short of being able to describe the behavior of the system. To obtain meaningful simulation results even in an environment with high ICT penetration, the ICT system has to be integrated into the simulation.

The term co-simulation refers to joint simulation of several aspects or domains of a system. Co-simulation takes under consideration complexity of the simulated system and influences between different aspects or domains interconnected in the same system. The term co-simulation was used to refer to hybrid hardware/software simulation systems, as defined in [1], [2], but it can also refer to a set of interconnected software simulations. In this work we distinguish between hardware/software co-simulation, in specific cases referred also to as hardware-in-the- loop [3], and purely software co-simulations.

Recent work on power system software co-simulation includes efforts to include communication network simulation [4]–[6] and the interaction with electricity markets [7]. An approach to simulate several domains in one tool is presented in [8], investigating influence of user behavior on power consumption in residential houses.

A multi-agent approach to design applications for the power system can be used to test distributed algorithms for intelligent control [9], monitoring and diagnostics [10], [11] or configuration of a power system [12]. In order to simulate advanced multi-agent control, monitoring or reconfiguration strategies and its impact on the power system, a specialized simulation environment is needed. The simulation tool can either offer many tools for representing complexity of a power system or be a flexible reconfigurable collection of simulation tools cooperating to express desired properties of a power system. In this work a co-simulation of multi-agent control and power system dynamics is explored.

II. Co-simulation composition

In order to compose a co-simulation out of required simulators, several steps need to be performed: identify required simulation tools (section II-B), identify and implement co-simulation requirements (section II-A), adjust simulators to work in a chosen co- simulation setup (section IV) and integrate simulators with an orchestrator (section V). This paper presents all steps of co-simulation integration for a chosen power system and multi-agent control simulation.

A. Co-simulation requirements

A co-simulation consisting of several separate simulations and models exchanging information, is required to: (1) track and progress simulation time – making sure that all simulations have access to current simulation time; (2) synchronize data exchange – synchronize simulation time in order to allow information exchange between simulations. The timing issue is very important in co-simulation resulting in realistic representation of correlations between simulations and interdependent domains; (3) facilitate data exchange- pass data between simulations; and finally (4) coordinate and schedule execution of all simulations. Depending on the co-simulation architecture, the realization of requirements (1)–(4) can be placed in the different entities in the co-simulation: in the simulations or the orchestrator. In case of the simulators, the functionality is

expected to be implemented in the simulation interface. The orchestrator should be designed to fulfill one or more of the co-simulation requirements, complementing the implementation of requirements in the simulation interfaces.

B. Software tools

In this paper a co-simulation set-up is composed of several software tools. In this work we have investigated integration of open-source software tools and their usability for a co-simulation set-up.

1) Mosaik: Mosaik is a simulation compositor and a powerful scenario specification framework. Mosaik aims to integrate many kinds of simulation programs and therefore offers a *network API* via which these programs can communicate with mosaik – irrespective of their programming language [13]. In this paper mosaik is used to integrate a control strategy and power system simulators in a single smart grid scenario. The process of combining external third party simulation platforms with an integration framework such as mosaic is described and the effort is evaluated. In this paper we investigate usability of mosaik in version 1.

2) Multi-agent platform: A multi-agent approach to design smart grid control strategies was used in this work. Popular MAS platforms are Jade [14], JDE [15], Jason [16] offering formalized approach to model and design agents, run-time environment and communication capabilities. The control strategy is simulated in Jade, popular multi-agent platform using standardized IEEE Computer Society FIPA (The Foundation for Intelligent Physical Agents) communication [17].

3) IPSYS: The power system is simulated in the open source multi-domain simulator IPSYS [18]. IPSYS is built around a quasi-static, fixed-time step energy system model and is intended as a simulator for distributed power systems. It features flexible configuration, system layout and control. IPSYS explicitly models the interaction between an electrical system and other, interconnected energy balances, such as district heating and water supply systems. In order to enable accurate model ling of the supervisory control of the system, the IPSYS model is designed to work with time step sizes down to a few seconds.

III. Co-simulation orchestration in mosaik

The mosaik framework is an integrative co-simulation platform. Mosaik allows a flexible approach towards co-simulation composition and offers a powerful modeling and specification language to automate the process of reusing existing models and platforms in orchestrated large-scale smart grid co-simulations.

In mosaik, simulation programs need to provide a *self description*. Combined with a *semantic meta-model*, this allows to create small and large scale scenarios, ranging from only a handful of simulated entities to multiple thousands [19]. The simulation of these scenarios involves two steps: *composition* and *execution* (or *stepping*). During the first phase, mosaik analyses the scenario, determines which simulation programs need to be started and how they should be parametrized. It also connects the simulators' model instances using the *ports* defined in their self-description. For example, a PV module may have a port for active and reactive power feed-in that can be connected to a PQ-port of a node in the power grid.

Based on the information gathered in these steps, mosaik creates a *schedule* that defines in which order the simulators need to be stepped during the execution phase (the actual simulation). Mosaik sends *step* commands to the simulators according to the schedule and exchanges data between the simulators. It also saves this data to an HDF5 database for later analysis.

A. Scheduling

Usually, the simulators involved in a co-simulation have dependencies between each other. Examples may be a PV simulation which requires input from a weather simulation or a power system simulation which requires the loads and feed-in from PV, wind or consumer simulations. Furthermore, the simulators may have different step sizes as shown in figure 1.

Fig. 1. An example of a co-simulation involving three simulators: *Sim1*, *Sim2*, and *Sim3*, with a step size of 30, 15, 60 seconds, respectively.

To resolve the dependencies and cope with varying step sizes, mosaik builds a directed acyclic graph *schedule graph* which defines the order in which the simulators are being started and stepped. Figure 2 shows the schedule graph for the example above assuming that *Sim2* provides input for *Sim1*.

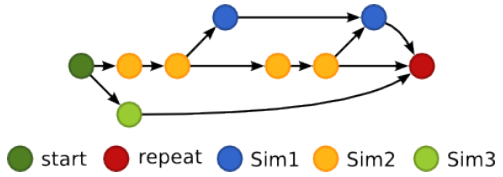

Fig. 2. The schedule graph for the example above, where *Sim1* depends on data provided by *Sim2*. When all simulators reach the red circle, mosaik jumps back to the beginning and repeats this process until the simulation ends.

B. Synchronization point

Let's consider a set of discrete-event simulations in a co-simulation $C = \{S_0, S_1, \ldots, S_n\}, n \in \mathbb{N}_0$, where the set of all fixed simulation steps is $P_C = \{p_{S_0}, p_{S_1}, \ldots, p_{S_n}\}, n \in \mathbb{N}_0, p \in \mathbb{N}_0$. Let Γ be a set of non-coprime numbers $\forall a, b \in \Gamma_C : gcd(a, b) \neq 1$, where $gdc : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ is greatest common divisor. Let's assume that P_C is a subset of Γ, therefore $P_C \subset \Gamma$ and is defined as:

$$\forall a, b \in P_C : gcd(a, b) \in P_C \tag{1}$$

Let Λ_C be a set of all schedules to execute simulations from set C in parallel. Synchronization point in a co-simulation is the simulation time where all simulations are synchronized. Let γ be the smallest synchronization point of simulations in C, $\gamma_C : \Lambda_C \rightarrow \mathbb{N}_1$. If P_C defined as in (1), then

$$\forall \lambda \in \Lambda_C : \gamma_C(\lambda) = \max(P_C) \quad (2)$$

where $\max : \mathbb{R}^n : \mathbb{R}, n \in \mathbb{N}_1$ is a maximum function, which is the largest value of a set. The equation (2) translates to statement that for all schedules in co-simulation C the smallest synchronization point is equal to the longest simulation step, if the simulation steps are described as in equation (1). The global synchronization point calculated as in (2), local synchronization point is chosen form only a subset of simulations from co-simulation, and can be calculated similarly. Let C' be the subset of simulation sin the co-simulation, $C' \in C$, and $P_{C'} = \{p_{S_0}, p_{S_1}, \ldots, p_{S_m}\}, m < n, m \in \mathbb{N}_0, p \in \mathbb{N}_0$.

In the example in figure 1 the global synchronization for simulation set $C = \{Sim1, Sim2, Sim3\}$ point is $\gamma_C = 60sec$ simulation time, where all simulators have results to share. For co-simulation set $C' = \{Sim1, Sim2\}$ the local synchronization point $\gamma_{C'} = 30sec$.

In mosaik, as presented in figure 2 the red circle is called a *global synchronization point* because the simulation time of all simulators is the same. However, not all participating simulators may be accessible to a control strategy or MAS, so mosaik introduced the concept of *public simulators*. Public simulators may have their own synchronization point with an interval smaller or equal than that of the global synchronization point. If *Sim1* and *Sim2* where public simulators, their synchronization point would be reached every 30 seconds. That means that control strategies can get active every 30 seconds instead of every 60 seconds.

IV. Adapting tools for co-simulation

In order to compose a co-simulation ecosystem out of available simulations, several modifications need to be performed. All simulators need to follow the overall design of the co-simulation implied by mosaik, to produce control signals arriving to all simulators at the desired time. In mosaik framework simulators are required to be discrete-event with fixed time steps. Mosaik uses fully static scheduling, every aspect of the schedule is determined before run time, therefore simulation need to define their fixed time step at the configuration time. Following this approach both control simulator and power system simulator (IPSYS) need to be discrete-event with fixed time step. The discrete event simulation (DES) performs actions in the step manner, where every event occurs in one instance of time, simulation step, and exchange the state of the system with other simulators and synchronize on discrete time steps.

In this section the process of integrating chosen tools into co-simulation framework is presented. Real-time control implementation in Jade need to be changed into discrete-event simulation, tackling timing issues, executing discrete time steps on a multi-agent system. Continuous time simulator IPSYS is also modified to become a discrete-event simulator with added functionality to modify variables externally.

A. Multi-agent system simulation

Multi-agent system (MAS) platforms are designed to execute and represent interactions between agents in real-time. In order to integrate a multi-agent system into a co-simulation framework, a real-time simulator need to be approximated with discrete event simulator, and the real time need to be dissected into time steps. In this work we present two solutions form MAS: time variant and time invariant control models.

1) DE time invariant solution: The multi-agent simulator MasSim was developed in RTLabOS project [20]. The RTLabOS project investigates software tools and platforms targeted at supporting smart grid research. This paper investigates how to decouple controller design and deployment from the chosen power system simulation and how different control strategies can be evaluated and compared using the same experimental setup (in software or hardware) as a basis for comparison. The MasSim simulator is designed to execute a single step of a simulation and store the result of the previous step locally. The simulation is event-based, agents are triggered when they receive a message, there is no local time considered. The real-time agent execution time is assumed to be no more then the simulation step size. An execution of a simulation time step can be rejected if the MasSim simulator assumes that the real-time execution takes more time than the simulation step.

The main issue of translating a multi-agent real-time system into a simulation is stopping the agent operation until the next simulation step is requested. In the MasSim approach most of parallel or sequential behaviors in the agent are event-based and implemented with use of *OneShotBehaviour* and *CyclicBehaviour* from Jade, therefore an agent is in an idle state until it receives a specific message. Time triggered continuous behaviors, represented as *TickerBehaviour* and *WakerBehaviour* in Jade, are excluded form MasSim implementation. In this case the multi-agent system does not progress its own time, it is informed about the current time and the time step from the execution co-simulation orchestrator.

The timing issues are ignored within the control model simulation step execution. The responsibility to keep track and propagate time is assigned to MasSimSlave agent, as shown in figure 3, and this entity does not progress the local time within the simulation time step requested form the co-simulation orchestrator. This simplification of timing brings ease of implementation: does not constrain the agent operation or computation to specific time and guarantees that a step will always produce an output, abstracting away the complexity of calculations and negotiations. This solution also brings limitations by removing a global source of a granular time, that can drive the realistic execution of agents, including computation and communication time, implementing timeouts and alarms.

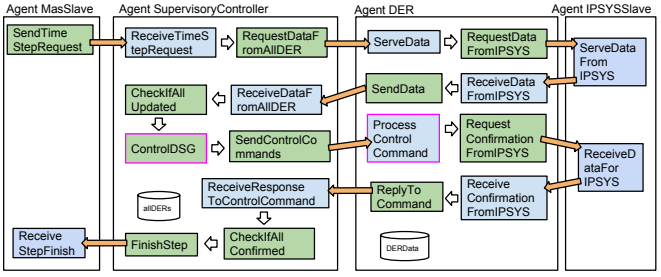

Fig. 3. Agents and participating in the implementation example in the MasSim simulator.

To trigger a single simulation step MasSimSlave agent is started through MasSim co-simulation interface. The MasSimSlave configuration specifies which agents need to be triggered, e.g. in figure 3 only the SupervisoryController agent is informed that an execution of a new step is requested. In the example in figure 3 the SupervisoryController agent is designed to request data from all known DERs, compute a control command, send the command to the DERs and return a success flag to the MasSimSlave agent.

2) DE time variant solution: In the SmartNord project the Jade framework has been chosen to implement the multi-agent control strategy (JadeSim). As outlined in the previous section Jade is tightly bound to real time (e.g., wall clock time). For example, if an agent needs to block its execution for 1 second, Jade will resume the agent after one second of real time has passed. In a simulation environment an arbitrary amount of simulation time (depending on the computational expense) may pass in one second of real-time. Using Jade in a simulation environment will result in time synchronization issues and possibly unpredictable behavior.

An unconstrained utilization of Jade is a requirement in the SmartNord project. All behaviors (including the time triggered *TickerBehaviour* and *WakerBehaviour*) must be supported. The same holds for Jade protocols, of which some also reference time in terms of deadlines (e.g. *ContractNet*).

To circumvent time synchronization issues, the classes of JADE subject to those issues were overridden where possible and replaced where not. Additionally, a new custom agent container was implemented which manages the simulation time. Protocols and behaviors were modified to request the current time from this container.

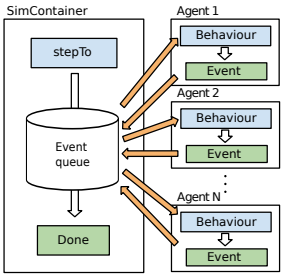

Fig. 4. Handling of Jade events in Smart Nord.

The custom *SimContainer* is able to advance simulation time by executing the events, as shown in figure 4. SimContainer provides a new method called *stepTo* which advances the simulation time to a given point in time. This is done by fetching and executing the next event from the event queue. Message events will insert a message into the receiving agents buffer and call the appropriate behavior, as it is done by Jade itself too. Timeout events are handled in the same way, albeit without inserting a message. The behaviors will in turn send new messages or block their execution, thereby creating new events. These events are inserted into the SimContainer event queue thereby closing the cycle.

B. IPSYS-DE

The IPSYS tool was originally designed as a self-contained simulator and includes a framework for the internal simulation of controllers as well as its own time generator. IPSYS-internal controllers, however, are limited to running once per simulation time step and are therefore not suitable for applications with discrete-event control as used here. To facilitate the integration into mosaik, two remote interfaces to IPSYS were developed: An interface to set and advance the simulation time step (remote control interface), and an interface to provide external access to the public part of the simulation state space by exposing all IPSYS ”sensor” and ”actuator” access points (state space interface). Both interfaces use XML-RPC to ease interoperability across platforms and programming languages.

V. Integration of Simulators with Mosaik

The two different ways to connect discrete-event control simulator with a power system simulator with use of mosaik are presented in this paper. The first approach to co-simulation treats controls strategy as a overall control of all simulated domains, gathering all information form simulators and models and controlling one or several domains. The second approach to co-simulation is to treat control strategy simulator as one of simulated domains and execute it simultaneously with other domains, here called a separate domain approach.

A. Overall control approach

In the overall control approach the control strategy is implemented as a separate simulator, outside of the co-simulation and supervises all running simulations.

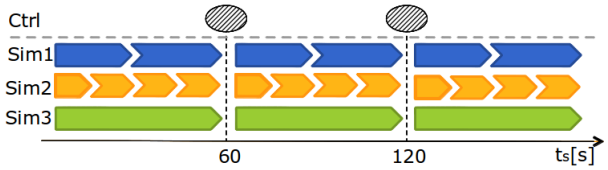

Fig. 5. Co-simulation scheduling in the overall control approach.

In this approach time flows from event to event for the control agents. Data from simulators may be requested at any point in time, but is only delivered at synchronization points. Figure 5 shows two synchronization points at 60 resp. 120 seconds of simulation time. At these points coherent simulation data is available from all simulators. A drawback of this approach is that the control strategy may only influence the simulations in the global synchronization point.

B. Separate domain approach

Separate domain approach to co-simulation of control and power system is to treat the implementation of the control as one of simulated domains and execute it simultaneously with other domains. In mosaik it is possible to specify the control strategy as a simulator, defining interfaces and data exchange and by implementing a proxy extending the mosaik Simulation API and designing data exchange ports specified in mosaik scenario composition.

As presented in example of a co-simulation composition in figure 6, *Ctrl* simulator can synchronize on data exchange with other simulations: *Sim1*, *Sim2* and *Sim3*. Depending on the specification of the mosaik scenario and defined ports for data exchange, *Ctrl* simulator can exchange data with *Sim1* and

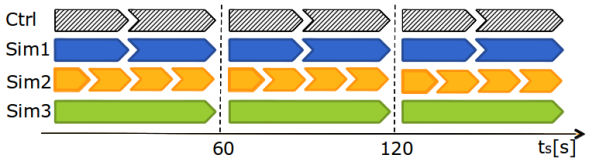

Fig. 6. Co-simulation scheduling in the separate domain approach.

Sim2 at the local synchronization point $\gamma_{\{Ctrl,Sim1,Sim2\}} = 30, 60, 90, \ldots [sec]$, and exchange data with all simulations at the global synchronization point $\gamma_{\{Ctrl,Sim1,Sim2,Sim3\}} = 60, 120, 180, \ldots [sec]$.

The advantage of this solution is an ease of adding a control strategy simulator into a co-simulation framework. With well specified mosaik ports, simulation can exchange data with other simulators, several control strategies can be added to one co-simulation scenario.

VI. Implementation

Th MasSim control simulator was tested together with separate domain approach co-simulation composition approach, JadeSim approach was tested together with overall control strategy co-simulation composition technique. This section presents the implementation details of APIs and interfaces between mosaik, MasSim, JadeSim and IPSYS-DE.

A. MasSim and IPSYS co-simulation set-up

Figure 7 presents a software architecture for control strategy co-simulation with separate domain approach, as presented in section V-B. The architecture consist of *MasSim Jade* simulator of a control strategy, *MasSim Proxy* implementing APIs to communicate with the simulator and interface for the co-simulation slave communicating with mosaik. Similarly IPSYS-DES is provided with *IPSYS proxy* to communicate with mosaik with use of the Simulation API.

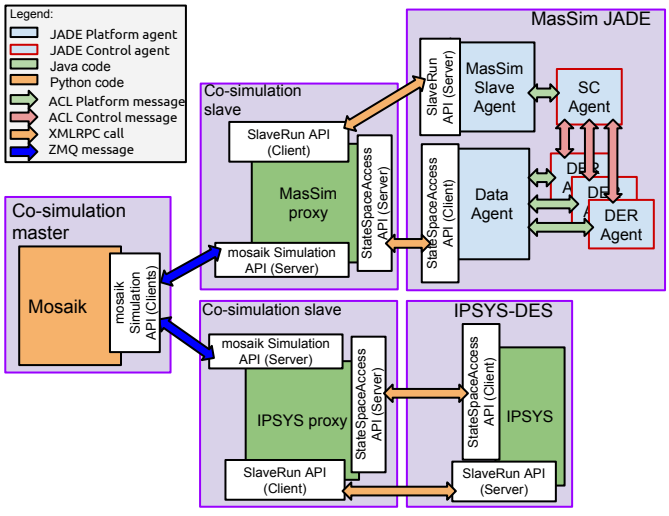

Fig. 7. Co-simulation architecture in the separate domain approach.

The data flow and co-simulation command flow are decoupled both in the MasSim and IPSYS-DE interface design. In the MasSim the MasSimSlave, using SlaveRun API, is responsible to execute a simulation step and Data agent is responsible to request, with use of StateSpaceAccess API, and host the data request service for all agents. The data kept in the Data agent is a representation of the state of other simulations in the co-simulation environment. Similarly the SlaveRun API is used to trigger IPSYS-DE and StateSpaceAccess API is used to request and send data to IPSYS-DE power system simulator. These two data streams are combined and handled by MasSimProxy and mapped to mosaik Simulation API.

B. JadeSim and IPSYS co-simulation set-up

Figure 8 shows the architecture for the overall control approach. The co-simulation slave *IPSYS proxy* is reused from the separate domain approach but the control simulation has been replaced by Jade SimContainer which communicates with mosaik using the control strategy (CS) API.

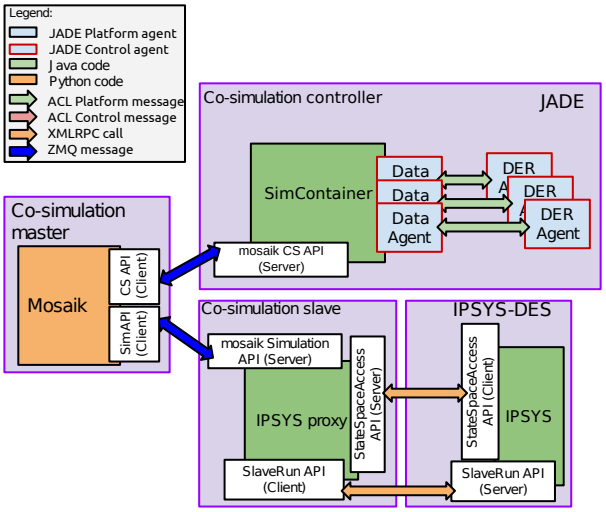

Fig. 8. Co-simulation architecture in the overall domain approach.

The SimContainer manages the execution of the Jade agents. It is therefore responsible to execute a simulation step (e.g. calling *stepTo*). Additionally, the container needs to push data into and pull data from the control strategy to the simulation models on each synchronization point. This is done using special agents called data agents, which act as interfaces to the simulated entities.

C. Interfacing MasSim and IPSYS-DE

APIs to interface IPSYS-DE and MasSim were generalized for any simulation in co-simulation and consist of API for simulator control *SlaveRun* and API for data retrieving *StateSpaceAccess*. Presented APIs are based on client-server design pattern, methods implemented in the server can be invoked by a server using XML-RPC call.

1) SlaveRun API: consist of several methods enabling managing and monitoring a simulation: *getScenarioStartTime* returns the start time of the scenario run in the simulator; *getScenarioEndTime* returns the end time of the scenario run in the simulator; *getScenarioTimestep* returns the time step of the scenario run in the simulator; *initializeSimulation* initializes the simulation and prepare to receive a simulation time-step;

doTimestep is invoked with a parameter specifying simulation time-step, starts a single and specific time-step in the simulation; *getLastExecutedTimestep* returns last successfully executed time-step.

2) StateSpaceAccess API: proposes following methods to retrieve the state space data from the simulator: *listAllStateVariables* returns a list of all the state variable names available form the simulator. There are two methods to query a specific variables: by name: *listStateVariable* and by a regular expression (regex) search pattern *listStateVariablesbyRegex*. To retrieve data from a simulation one of two methods can be used: to get a list of all state variables *getAllStateVariables* is used, in order to get a variable with a specific name *getStateVariables* with a name argument is used. To set a variable *setActuator* is used with a full name argument (which for some simulators can be a set of names) and a value that should be set.

D. Interfacing mosaik

In its current version, mosaik differentiates between simulators and control strategies (CS). Simulators use a push-based API, that is, mosaik pushes data to the simulators, requests them to step and asks for their new state. CS on the other hand request data from mosaik and ask it if they can step. Mosaik only answers these requests.

Apart from that, both APIs are very similar. The APIs are based ZeroMQ sockets and send JSON-serialized messages. The communication between simulators/CS and mosaik always starts with an *init* message to set basic parameters and the model configuration. After that, *static data* (data that doesn't change over time) and *entity relations* (which entities are connected to which) are exchanged. This is followed by the *stepping* phase were mosaik repeatedly *sets* input data to the simulators, requests them to *step* and *gets* the new state information from them. CS request data from mosaik in that phase, perform their calculations and, if they are done, notify mosaik that they are ready for the next *step*.

The low-level protocol for the messages exchanged is documented in [21]. To ease the implementation of the mosaik API, high-level interface implementations are accessible, currently only available for Python and Java. These interfaces provide an abstract base class (Python) or an interface (Java) that maps the messages of the low-level API to function calls. The high-level API also sets up the required network sockets and implements a simple event loop.

VII. Experimental results

In order to validate the two approaches to co-simulating power system and multi-agent control, a single scenario was developed to compare results obtained from two experiments.

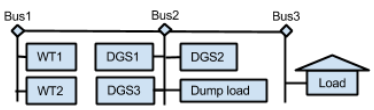

Fig. 9. Power system use case represented in IPSYS.

Data statistics	BC/E1	BC/E2
Minimum	0	-9.004000
Maximum	0	9.909000
Mean	0	-0.003284
Standard deviation	0	0.003284

TABLE I. Data statistics for experiments 1 and 2.

In Experiment 1 co-simulation of MasSim and IPSYS-DE using mosaik is evaluated, Experiment 2 co-simulates JadeSim and IPSYS-DE using mosaik. Results from these to experiments are compared to IPSYS running autonomously. IPSYS-DE was verified to produce the same results as IPSYS.

The single scenario used to compare presented approaches is a 3 bus island system consisting of two wind turbines, three controllable diesel generators, a dump load and a residential load, as presented in figure 9. Diesel generator DGS1 is a grid forming unit, therefore it is required to run at all time. All diesel generators have a droop control to keep voltage and frequency within required limits. Diesel generators can be controlled externally with an on/off signal, and provide their on/of state, as well as active and reactive power measurements.

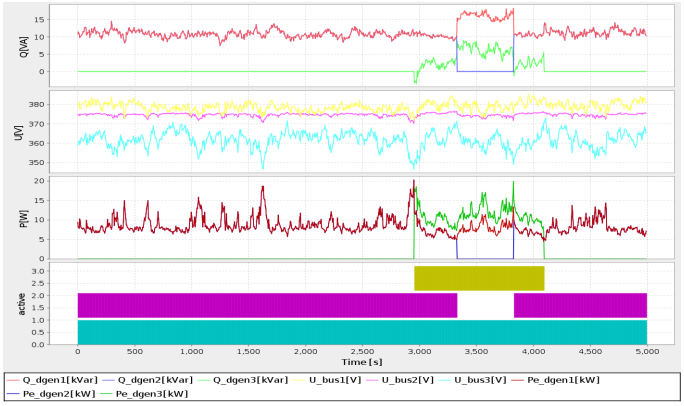

Fig. 10. IPSYS simulation output of a presented power system scenario.

The scenario is designed to control the dispatch of three power-balancing diesel generators in the isolated grid shown in figure 9. The results are presented in figure 10.

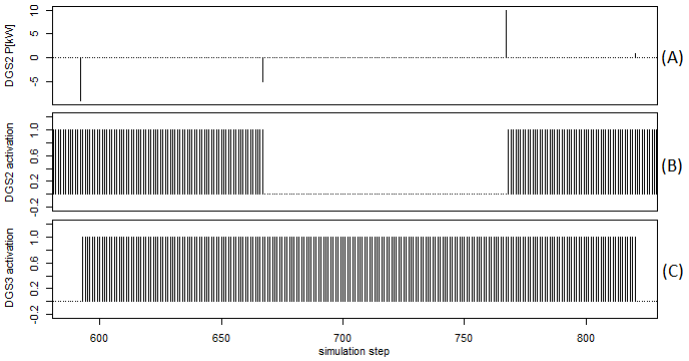

Fig. 11. Error in active power of DGS2 in experiment 2 and its correlation to switching time of DGS2 and DGS3.

The difference between experiment 1 and 2 compared to base case is presented in table I. Table I presents data statistics

for difference between DGS2 active power in kW obtained from experiments (E1 and E2) and base case (BC). The co-simulation of MasSim and IPSYS-DE using mosaik produced results identical with the base case run in IPSYS. However co-simulation of JadeSim and IPSYS-DE using mosaik produced error compared with the base case. The error produced by this setup has a standard deviation of *0.003284*. The comparison of active power at DGS2 and switching times of DGS1 and DGS2 is presented in figure 11. The error in operation of DGS2 is correlated to switching time of DGS2 and DGS3, co-simulation form experiment 2 is delayed one step to activate or deactivate controllable units. The cause for this late switching are the synchronization points mentioned in V-A. The control strategy can only observe the current state at a synchronization point. But changes to that state like issuing a switch can only take place in the time frame after the synchronization point, resulting in a delayed switch. The late switching affects DGS2 active power consumption, as shown in figure 11, and late switching of DGS3 affects DGS2 which has to take over some of the extra load.

VIII. Conclusion

In this paper describes a process of integrating a multi-agent control simulator and power system dynamic simulator into a single smart grid co-simulation. Two models of control designed in a multi-agent are described: time variant and time invariant models are presented and evaluated. Next two methods of integrating presented simulators with mosaik tool and power system dynamics simulator IPSYS are presented and verified with experimental results. The two multi-agent control simulations, MasSIm and JadeSim were verified. As shown in section VII the same behavior of controlled units, in comparison to base scenario run in a single simulation environment IPSYS. In the experiments with co-simulation composition the separate domain approach produced no errors, and overall control approach produced errors due to late switching associated with problems with synchronization of data exchange.

References

[1] D. Becker, R. K. Singh, and S. G. Tell, "An engineering environment for hardware/software co-simulation," in *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*. IEEE, 1992, pp. 129–134.

[2] N. S. Voros, L. Sánchez, A. Alonso, A. N. Birbas, M. Birbas, and A. Jerraya, "Hardware/software co-design of complex embedded systems: an approach using efficient process models, multiple formalism specification and validation via co-simulation," *Design Automation for Embedded Systems*, vol. 8, no. 1, pp. 5–49, 2003.

[3] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[4] H. Lin, S. Sambamoorthy, S. Shukla, J. Thorp, and L. Mili, "Power system and communication network co-simulation for smart grid applications," in *Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES*. IEEE, 2011, pp. 1–6.

[5] A. T. Al-Hammouri, M. S. Branicky, and V. Liberatore, "Co-simulation tools for networked control systems," in *Hybrid Systems: Computation and Control*. Springer, 2008, pp. 16–29.

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

[7] G. Conzelmann, G. Boyd, V. Koritarov, and T. Veselka, "Multi-agent power market simulation using emcas," in *Power Engineering Society General Meeting, 2005. IEEE*. IEEE, 2005, pp. 2829–2834.

[8] A. Kashif, J. Dugdale, and S. Ploix, "Simulating occupants' behavior for energy waste reduction in dwellings: A multiagent methodology," *Advances in Complex Systems*, vol. 0, no. 0, p. 1350022, 2013.

[9] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation," in *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*. IEEE, 2009, pp. 1–8.

[10] E. Davidson, S. McArthur, J. R. McDonald, T. Cumming, and I. Watt, "Applying multi-agent system technology in practice: automated management and analysis of scada and digital fault recorder data," *Power Systems, IEEE Transactions on*, vol. 21, no. 2, pp. 559–567, 2006.

[11] S. D. J. McArthur, S. Strachan, and G. Jahn, "The design of a multi-agent transformer condition monitoring system," *Power Systems, IEEE Transactions on*, vol. 19, no. 4, pp. 1845–1852, 2004.

[12] A. M. Kosek, O. Gehrke, and D. Kullmann, "Fault tolerant aggregation for power system services," in *IEEE International Workshop on Intelligent Energy Systems (IWIES3013), 39th Annual Conference on IEEE Industrial Electronics Society*, November 2013.

[13] S. Schütte, S. Scherfke, and M. Sonnenschein, "mosaik – smart grid simulation api," in *Proceedings of SMARTGREENS 2012 – International Conference on Smart Grids and Green IT Systems*, April 2012.

[14] "JADE: Java Agent DEvelopment Framework," 1999. [Online]. Available: http://jade.tilab.com/

[15] "JACK Intelligent Agents Development Environment," Agent Oriented Software Pty. Ltd., Tech. Rep., 2010.

[16] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. Wiley. com, 2007, vol. 8.

[17] "FIPA ACL Message Structure Specification," *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*, 2002.

[18] H. Bindner, O. Gehrke, P. Lundsager, J. C. Hansen, and T. Cronin, "Ipsys–a tool for performance assessment and supervisory controller development of integrated power systems with distributed renewable energy," *Risø National Laboratory*, 2004.

[19] S. Schütte and M. Sonnenschein, "mosaik – scalable smart grid scenario specification," in *Proceedings of the 2012 Winter Simulation Conference*, December 2012.

[20] "RTLabOS: Smart Grid Laboratory Operating System for Real-time Control, ForskEL project, Denmark," 2013-2014.

[21] S. Scherfke and S. Schütte, "mosaik – Architecture Whitepaper," OFFIS – Institute for Information Technology, Tech. Rep., October 2012.